# EFFICIENT TRAJECTORY AND POLICY OPTIMIZATION USING DYNAMICS MODELS

A Dissertation
Presented to
The Academic Faculty

By

Xinyan Yan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Robotics

School of Interactive Computing
Georgia Institute of Technology

August 2020

**EFFICIENT TRAJECTORY AND POLICY OPTIMIZATION USING DYNAMICS MODELS**

Thesis committee:


Dr. Byron Boots
Paul G. Allen School of Computer Science and Engineering
*University of Washington*

Dr. Nathan Ratliff
NVIDIA's Robotics Lab
*NVIDIA Corporation*

Dr. Sonia Chernova
School of Interactive Computing
*Georgia Institute of Technology*

Dr. Vikas Sindhwani
Google Brain
*Google LLC*

Dr. Frank Dellaert
School of Interactive Computing
*Georgia Institute of Technology*



Date approved: June 29, 2020

Theory is the first term in the Taylor series expansion of practice.

*Thomas M. Cover*

For my mom and my wife

## ACKNOWLEDGMENTS

I would like to thank the many people who have supported me and believed in me throughout my study at Georgia Tech. Without them, I could not have completed this long journey successfully.

First and foremost, I wish to express my deepest gratitude to my advisor Byron Boots. He inspired and guided me to be an independent robotics researcher, and provided tremendous support and invaluable advice for setting and reaching my goals even when the road got tough. He was there for me all the time. Without his guidance and impact in my life, I would not have pursued this Ph.D. program and none of work in this thesis would have been possible.

I am indebted to my exceptional thesis committee members for their help in preparation of this work: Sonia Chernova, Frank Dellaert, Nathan Ratliff and Vikas Sindhwani. Their insightful questions widened my research perspective and their wise suggestions were priceless in developing my career. I also am sincerely grateful for receiving mentorship from Scott Klasky and Ellen Yi-Luen Do, who shaped my view as a researcher during my very early years at Georgia Tech and have been very supportive ever since.

I am particularly fortunate to have interned at several amazing places. The experiences are so valuable that had transformed my perspective towards robotics. I would like to acknowledge my mentors during the internships: Krzysztof Choromanski, Vikas Sindhwani, Aleksandra Faust, James Davidson at Google, David Ilstrup, Aniket Murarka at Honda Research Institute, and Jason Chen-Chi Chu, Wenyi Zhao at Daqri.

I am extremely lucky to be a graduate student in Robot Learning Lab and Institute for Robotics and Intelligent Machines at Georgia Tech. The collaborative and interdisciplinary atmosphere allowed me to explore tremendous research opportunities. I would like to especially thank Ching-An Cheng for carefully listening to my preposterous ideas and providing helpful critics and suggestions that significantly improved my essential skills as a

researcher, and Yunpeng Pan and Bo Xie for introducing me to interesting research areas and taught me when my background was lacking. Many thanks are due to my collaborators and lab-mates whom I learned from immensely over the years: Evangelos Theodorou, Vadim Indelman, Le Song, Mustafa Mukadam, Nolan Wagener, Jing Dong, Anqi Li, Sasha Alexander Lambert, Jacob Sacks, Zhaoyang Lv, Keuntaek Lee, Kamil Saigol, Munzir Zafar, Peter Vieira, and Chih-Pin Hsiao.

I would like to pay my special regards to Mike Stilman who simulated my passion for robotics, and Karsten Schwan who had strong confidence in me. I miss their vision, spark, and wisdom.

Finally, I would like to thank my family that always has faith in me during all the ups and downs.

**TABLE OF CONTENTS**

**Acknowledgments** . . . . . v

**List of Tables** . . . . . xii

**List of Figures** . . . . . xiii

**List of Acronyms** . . . . . xviii

**Chapter 1: Introduction** . . . . . 1

1.1 Main Contributions . . . . . 2

**I Incremental Trajectory Optimization** **5**

**Chapter 2: Incremental Trajectory Estimation and Mapping** . . . . . 6

2.1 Introduction . . . . . 6

2.2 Batch GP-based Trajectory Estimation and Mapping . . . . . 8

2.2.1 GP Representation of Trajectories . . . . . 8

2.2.2 Landmarks and Measurements Models . . . . . 10

2.2.3 Estimation of Trajectory States and Landmark Locations . . . . . 10

2.2.4 State Interpolation . . . . . 13

2.2.5 Sparse Gaussian Process Regression . . . . . 13

2.3 Batch GP-Regression with Variable Reordering . . . . . 15

2.4 Fast Incremental Updates . . . . . 17
2.4.1 The Bayes Tree Data Structure . . . . . 18
2.4.2 Faster Updates Through Interpolation . . . . . 21
2.5 Experimental Results . . . . . 23
2.5.1 Synthetic SLAM Exploration Task . . . . . 26
2.5.2 Autonomous Lawnmower . . . . . 29
2.5.3 Victoria Park . . . . . 31
2.6 Conclusion . . . . . 32

**Chapter 3: Prediction under Uncertainty for Model Predictive Control . . . . . 34**

3.1 Introduction . . . . . 34
3.2 Sparse Spectral Representation of GPs . . . . . 35
3.3 Prediction under Uncertainty . . . . . 38
3.3.1 Exact Moment Matching (SSGP-EMM) . . . . . 39
3.3.2 Linearization (SSGP-Lin) . . . . . 43
3.4 Model Predictive Control under Uncertainty . . . . . 44
3.4.1 Problem Setting . . . . . 45
3.4.2 MPC via Probabilistic Trajectory Optimization . . . . . 45
3.4.3 Gaussian Belief Space Dynamics using SSGP . . . . . 47
3.4.4 Comparison with Other Methods . . . . . 47
3.5 Experimental Results . . . . . 48
3.5.1 Computational Efficiency . . . . . 50
3.5.2 Accuracy of Multi-Step Prediction . . . . . 50

3.5.3 Model Predictive Control . . . . . 51
3.6 Conclusion . . . . . 53

**II Unbiased Policy Optimization 55**

**Chapter 4: Predictable Online Policy Optimization . . . . . 56**
4.1 Introduction . . . . . 56
4.2 Problem Definition . . . . . 58
4.3 IL and RL as Predictable Online Learning . . . . . 58
4.3.1 IL as Online Learning . . . . . 59
4.3.2 RL as Online Learning . . . . . 60
4.3.3 Predictability . . . . . 62
4.4 Predictor-Corrector Learning . . . . . 62
4.4.1 The PICCOLO Idea . . . . . 63
4.4.2 The Meta Algorithm PICCOLO . . . . . 64
4.4.3 Summary: Why Does PICCOLO Work? . . . . . 67
4.5 Theoretical Analysis of PICCOLO . . . . . 68
4.5.1 Convergence Properties . . . . . 69
4.6 Extending PICCOLO to Nonlinear Loss Functions . . . . . 70
4.6.1 Mirror Descent . . . . . 70
4.6.2 Follow-the-Regularized-Leader . . . . . 71
4.7 Experiments . . . . . 73
4.7.1 Linear PICCOLO on RL Tasks . . . . . 73
4.7.2 Nonlinear PICCOLO on IL Tasks . . . . . 76

4.8 Conclusion . . . . . 77

**Chapter 5: Explaining Fast Improvement in Online Policy Optimization . . . . 79**

5.1 Introduction . . . . . 79

5.2 Background: Online Policy Optimization . . . . . 82

5.2.1 Policy Optimization . . . . . 82

5.2.2 Reducing Policy Optimization to Online Learning . . . . . 82

5.2.3 Performance Guarantees in Online Policy Optimization . . . . . 84

5.3 Bias-Dependent Convergence Rates . . . . . 86

5.3.1 Setup and Assumptions . . . . . 86

5.3.2 Convergence Rate in Expectation . . . . . 87

5.3.3 Convergence Rate in High Probability . . . . . 88

5.3.4 Proof Sketch for Theorem 2 . . . . . 89

5.4 Case Studies . . . . . 90

5.4.1 Online Imitation Learning . . . . . 90

5.4.2 Interactive System Identification for Model-based RL . . . . . 91

5.5 Experimental Results . . . . . 92

5.6 Related Work and Discussion . . . . . 94

**Chapter 6: Trajectory-wise Control Variates for Policy Optimization . . . . . 96**

6.1 Introduction . . . . . 97

6.2 Problem Setup and Background . . . . . 98

6.2.1 Policy Gradient Methods: Pros and Cons . . . . . 99

6.2.2 Variance Reduction and Control Variate . . . . . 101

6.2.3 Control Variate Method and Difference Estimator . . . . . 101
6.2.4 Common Control Variates for Policy Gradient Methods . . . . . 102
6.3 Why We Need New Control Variates . . . . . 103
6.4 Trajectory-wise Control Variates . . . . . 105
6.4.1 A Divide-and-Conquer Strategy . . . . . 106
6.4.2 Design of TrajCV . . . . . 107
6.4.3 Algorithm Example . . . . . 109
6.4.4 Optimality of the Natural Ordering . . . . . 110
6.5 Experimental Results and Discussion . . . . . 112

**III Conclusion** **115**

**Chapter 7: Conclusion and Discussion** . . . . . 116
7.0.1 Future Directions . . . . . 116

**Appendices** . . . . . 119
Appendix A: Appendix for Chapter 3 . . . . . 120
Appendix B: Appendix for Chapter 4 . . . . . 129
Appendix C: Appendix for Chapter 5 . . . . . 167
Appendix D: Appendix for Chapter 6 . . . . . 184

**References** . . . . . 194

## LIST OF TABLES

2.1 Cost of Cholesky factorization with different ordering methods including ordering time. . . . . 17

2.2 Summary of the experimental datasets . . . . . 25

3.1 Examples of continuous shift-invariant positive-definite kernels and their corresponding spectral densities, where $r = \frac{\sqrt{2\nu}\|x-x'\|_2}{\ell}$, $K_\nu$ is a modified Bessel function, and $h = \frac{2^d \pi^{\frac{d}{2}} \Gamma(\nu+\frac{d}{2})(2\nu)^\nu}{\Gamma(\nu)\ell^{2\nu}}$. . . . . 43

3.2 Comparison of our proposed methods and GP-EMM [34, 35] in terms of computational complexity and generalizability. . . . . 45

3.3 Comparison of trajectory optimization-related methods with learned dynamics models. They include: our proposed algorithm, iLQG-LD [56], PDDP [41], AGP-iLQR [57], Minimax DDP [59] and SDDP with NN [58]. 48

5.1 Comparison of different learning settings. . . . . 94

B.1 Tasks specifics and hyperparameters. . . . . 165

## LIST OF FIGURES

1.1 The tradeoffs in using dynamics models that appear in incremental trajectory optimization and episodic policy optimization. (a) In incremental trajectory optimization, full model is often too computationally expensive. The objective is to develop methods that rely on approximate models in order to meet the real-time requirement while having a small performance loss. (b) In episodic policy optimization, using an imperfect model alone without data suffers from performance bias. The goal is to develop theories and algorithms that can leverage imperfect models to accelerate learning while avoiding performance bias. . . . . 3

2.1 Sparse information matrices. The information matrix $\mathcal{I}$ with XL ordering (a), and SYMAMD ordering (b). Both sparse matrices have the same number of non-zero elements, yet the second matrix can be factored much more efficiently due to the heuristic ordering of the matrix columns. (See Table 2.1). For illustration, only 200 trajectory states are shown here. . . . . 14

2.2 The Cholesky factors $\mathcal{L}$ of $\mathcal{I}$. In (a), $\mathcal{L}$ is computed with XL ordering, which exhibits fill-in. In (b), $\mathcal{L}$ is computed with SYMAMD ordering, which is more sparse. For illustration, only 200 states are shown here. . . . 17

2.3 The effect of interpolation and without interpolation. (a) Measurements are fully utilized. (b) Missing state, using interpolated measurements. (c) Missing state, ignore measurements. (d) Missing state, using sparse interpolated measurement. . . . . 20

2.4 Synthetic dataset: Ground truth, dead reckoning path, and the estimates are shown. State and landmark estimates obtained from BTGP approach are very close to ground truth. . . . . 25

2.5 Synthetic dataset: Comparison of the computation time of three approaches PB, PBVR, and BTGP. The modifiers /1 and /10 indicate frequency of estimate updates — the number of range measurements between updates. For example: *BTGP/1* updates the estimate after 1 new range measurement using BTGP. Likewise *BTGP/10* updates the estimate after 10 new range measurements using BTGP. Due to the large number of landmarks, 298, variable reordering dramatically improves the performance. . . . . 26

2.6 Synthetic dataset: Trade-off between computation time and accuracy if BTGP makes use of interpolation. The $y$-axis measures the RMSE of distance errors of the estimated trajectory states and total computation time with increasing amounts of interpolation. The $x$-axis measures the time step difference between two estimated (non-interpolated) states. The results indicate that interpolating $\sim 90\%$ of the states (i.e.,estimating only $\sim 10\%$ of the states) while running BTGP can result in a $33\%$ reduction in computation time over iSAM 2.0 without sacrificing accuracy. . . . . 27

2.7 Autonomous Lawnmower dataset: Ground truth, dead reckoning path and estimates are shown. The range measurements are sparse, noisy, and asynchronous. Ground truth and the estimates of path and landmarks obtained from BTGP are very close. . . . . 28

2.8 Autonomous Lawnmower dataset: Comparison of the computation time of PB, PBVR, and BTGP. As in Figure 2.5, /1 and /10 are modifiers — the number of range measurement between updates, and no interpolation is used by BTGP. The 'gap' in the upper graph is due to a long stretch around timestep 5000 with no range measurements . Due to the low number of landmarks, variable reordering does not help The incremental BTGP approach dramatically reduces computation time. . . . . 29

2.9 Autonomous Lawnmower dataset: Trade-off between computation time and accuracy if BTGP makes use of interpolation. The $y$-axis measures the RMSE of distance errors and total computation time with increasing amounts of interpolation. The $x$-axis measures the time step difference between two estimated (non-interpolated) states. The results indicate that interpolating $\sim 80\%$ of the states within BTGP results in only an 8cm increase in RSME while reducing the overall computation time by $68\%$ over iSAM 2.0. . . . . 30

2.10 Victoria Park dataset: Dead reckoning and estimated path obtained from BTGP approach. . . . . 31

2.11 Victoria Park dataset: Comparison of the computation time of three approaches PB, PBVR, and BTGP. As in Figure 2.5 and Figure 2.8, the modifiers /1 and /10 indicate frequency of state updates. Since many landmarks are involved, PBVR dramatically improves performance, compared to PB. The incremental BTGP algorithm improves performance even further. Unlike in previous datasets, we did not evaluate the trade-off between interpolation and accuracy for Victoria Park, since we do not have access to ground truth and cannot evaluate the effect on accuracy. However, like previous datasets, interpolation can greatly increase the speed of BTGP. . . 32

3.1 (a)-(b): Approximate inference accuracy test. The vertical axis is the squared error of cost predictions for (a) quadrotor system and (b) Puma 560 system. Error bars represent standard deviations over 10 independent rollouts. (c)-(d): Comparison of computation time on a log scale between (c) SSGP-Lin and GP-EMM; (d) SSGP-EMM and GP-EMM. The horizontal axis is the input and output dimension (equal in this case). Vertical axis is the CPU time in seconds. . . . . 49

3.2 PUMA-560 and quadrotor tasks . . . . . 52

3.3 Comparison of the drifting performance using 50 (left), 150 (middle) and 400 (right) random features. Blue lines are the solution provided in [61]. . 53

3.4 Cost comparison for arm and quadrotor tasks. In The results are the average over 3 independent trials. . . . . 53

3.5 The AutoRally car and the test track. . . . . 54

4.1 Performance of PICCOLO with different predictive models. $x$ axis is iteration number and $y$ axis is sum of rewards. The curves are the median among 8 runs with different seeds, and the shaded regions account for $25\%$ percentile. ADAM is used as the base algorithm, and the update rule, by default, is PICCOLO; e.g.,TRUEDYN in (a) refers to PICCOLO with TRUEDYN predictive model. (a) Comparison of PICCOLO and DYNA with adversarial model. (b) PICCOLO with the fixed-point setting (4.8) with dynamics model in different fidelities. BIASEDDYN0.8 indicates that the mass of each individual robot link is either increased or decreased by $80\%$ with probability 0.5 respectively. . . . . 75

4.2 Performance of PICCOLO in various tasks. $x$ axis is iteration number and $y$ axis is sum of rewards. The curves are the median among 8 runs with different seeds, and the shaded regions account for $25\%$ percentile. . . . . 75

4.3 Performance comparison of ADAM, DAGGER, and PICCOLO'ed DAGGER with a biased dynamics model. $x$ axis is iteration number and $y$ axis is sum of rewards (the greater the better). The curves are the median among 4 runs with different seeds, and the shaded regions account for $25\%$ percentile. BIASEDDYN indicates that the mass of each individual robot link and damping of revolute joint is either increased or decreased by $80\%$ with probability 0.2 respectively. And BIASEDDYN-FP is approximated by applying three update steps (4.10) by gather a new batch of data using dynamics model in each round. . . . . 77

5.1 The convergence rates of online IL when the policy class has zero bias (Figure 5.1a) and an additional bias due to an $\ell_2$-norm constraint on the weights (Figure 5.1b). The rates are obtained by fitting the curve of the average loss $\frac{1}{N}\sum_{n=1}^{N} l_n(\theta_n)$ with parametric $O(1/N)$ and $O(1/\sqrt{N})$ upper bounds to minimize $\ell_1$-error. The average loss curve is the median, and the shaded region represents $10\%$ and $90\%$ percentile, over 4 random seeds, due to the randomness in the initial state of the MDP and the initialization of the policy. . . . . 92

6.1 An illustration of the effects of state-action CV (6.5) and TrajCV (6.11). One row corresponds to one policy component $g_t = \rho_t c_{t:T}$, and thick borders indicate the random variables of which $g_t$ is a function. State-action CV reduces the variance due to the random variables in red, whereas TrajCV *additionally* affect the variance stemming from the random variables in green. 107

6.2 Components of $\mathrm{Tr}\left(\mathbb{V}[g_t]\right)$, for $t = 100$, evaluated at policies generated under the "upper bound" setting. $\sigma$ denotes the initial value of policy variance (defined in Theorem 5). The $x$-axis denotes the iteration number, and the $y$-axis is in $\log$ scale. The two vertical dashed lines mark the boundaries of iterations where the expected accumulated rewards is between $50$ and $900$. 112

A.1 KL divergences between SSGP-EMM and GP-EMM, SSGP-Lin and GP-Lin 128

B.1 The update rule, by default, is PICCOLO. For example TRUEDYN in (a) refers to PICCOLO with TRUEDYN predictive model. (a), (b): Comparison of PICCOLO and DYNA with adversarial model using NATGRAD and TRPO as base algorithms. (c), (d): PICCOLO with the fixed-point setting (4.8) with dynamics model in different fidelities. BIASEDDYN0.8 indicates that the mass of each individual robot link is either increased or decreased by $80\%$ with probability 0.5 respectively. . . . . 165

B.2 The performance of PICCOLO with different predictive models on various tasks, compared to base algorithms. The rows use ADAM, NATGRAD, and TRPO as the base algorithms, respectively. $x$ axis is iteration number and $y$ axis is sum of rewards. The curves are the median among 8 runs with different seeds, and the shaded regions account for $25\%$ percentile. . . . . 166

D.1 Bayes networks for the random variables in $g_t$ (6.2), before and after reparameterization. After policy is reparameterized, action $a_k$ is decided by state $s_k$ and action randomness $\xi_k$. . . . . 187

D.2 The exact same settings as Figure 6.3 except that the state-action CV and TrajCV are given by $\hat{Q}^{(\text{diff})}$ and $\hat{Q}^{(\text{diff-GN})}$ (Figure D.2a), and $\hat{Q}^{(\text{next})}$ and $\hat{Q}^{(\text{next-GN})}$ (Figure D.2b). . . . . 192

## SUMMARY

Data-driven approaches hold the promise of creating the next wave of robots that can perform diverse tasks and adapt to unstructured environments. However, gathering data of physical systems is often a labor-intensive, time-consuming, and even dangerous process. This issue of data scarcity motivates us to design algorithms that benefit from prior knowledge while avoiding relying too much on domain knowledge. One general and compact form of prior knowledge is dynamics models; they summarize our knowledge of the robot in the mechanical design and prior interactions with the robot through system identification. Unfortunately, often utilizing dynamics models to their full potential is not straightforward: *1)* they are computationally expensive, and *2)* they can even be harmful if the model errors are not taken into account. In this thesis, we address these two issues of using dynamics models by focusing on a central problem in robotics: trajectory and policy optimization. We develop new algorithmic and theoretic foundations of *1) computationally efficient* trajectory optimization and *2) unbiased sample efficient* policy optimization. Our research increases the practicality of continuous-time linear dynamics models and Gaussian process dynamics models in real-time incremental trajectory optimization, and accelerates policy optimization by utilizing dynamics models for prediction and control variates while avoiding performance bias due to model errors. We evaluate our approaches on a series of robot estimation, planning, and control tasks that involve both simulated data and real robotic systems.

# CHAPTER 1
# INTRODUCTION

Trajectory and policy optimization is a central problem in robotics. One unified view towards this problem is through the lens of stochastic optimization. Recently, in many challenging stochastic optimization problems, completely data-driven approaches that are based on deep learning have demonstrated impressive progress. Examples include computer vision, speed recognition, natural language processing, and playing chess and Go. However, this progress can not be translated to robotics directly, because the completely data-driven paradigm can only thrive on a huge amount of data, whereas in robotics gathering data of physical systems is a time-consuming and labor-intensive process.

This issue of data scarcity motivates us to design algorithms that can not only improve from data but also benefit from *prior knowledge*. One well-studied form of prior knowledge is dynamics models, which describe how the state of a robot evolves over time. Dynamics models are especially appealing as prior knowledge because they are both general and informative; they can capture task-independent information in a system-independent representation and provide crucial information about the robotic system for solving almost all robotics tasks.

Unfortunately, despite the aforementioned strengths of dynamics models as prior knowledge, designing algorithms that leverage dynamics models to their full potential is far from being straightforward. Two of the main difficulties of utilizing dynamics models are caused by the *computational burden* and the unavoidable *model errors*. Dynamics models can be too computationally expensive to use in robotic tasks that demand real-time and low-latency updates during execution, which is made even worse given limited onboard computing resources. Furthermore, the errors in dynamics model are inevitable; they can arise from many sources including nonstationarity of the system, model class bias, and error

in generalization due to a finite sample. These model errors can be amplified and reflected in final performance bias.

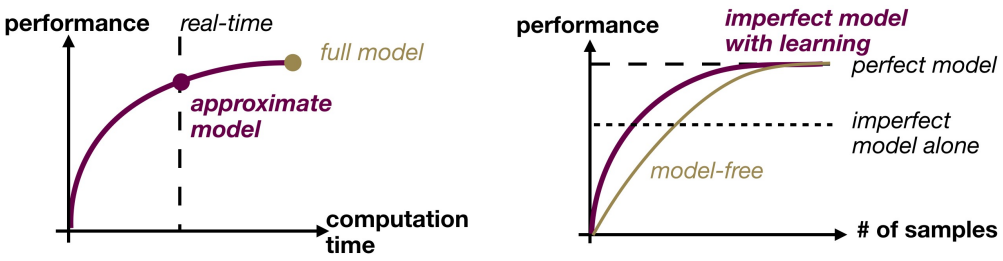
Prior to the work described in this thesis, stochastic continuous-time linear dynamics models have been used to generated a continuous-time prior distribution over trajectories for solving the trajectory estimation and mapping problem, and Gaussian process (GP) dynamics models have been used to realize state propagation in belief space for solving the model predictive control (MPC) problem under uncertainty. However, their high computational complexity restricts their use only to batch settings. Furthermore, in policy optimization, although model-based methods have been gaining lots of attention recently, most of them suffer from a common drawback: when the model is inaccurate, the performance of the policy can become biased away from the best achievable in the policy class.

To address these issue of computational bottleneck and performance bias, this thesis will focus on two ideas: sparse GP algorithms and predictable online learning, respectively. Based on these two ideas, novel algorithms and theories are developed to address the shortcomings and limitations of previous methods. Unlike full GP methods, sparse GP methods provides a desirable tradeoff between computation and performance: a large gain in computation can be obtained with a minor decrease in performance (illustrated in Figure 1.1a); Unlike most model-based policy optimization approaches, the predictable online learning framework provides a theoretical foundation of designing sample efficient yet unbiased model-based policy optimization algorithms (illustrated in Figure 1.1b).

## 1.1 Main Contributions

This thesis included the following major contributions:

- (Chapter 2) We developed a fast incremental algorithm for the simultaneous trajectory estimation and mapping (STEAM) problem for mobile robots that use stochastic continuous-time linear dynamics models [1, 2]. To speed up the solution time, we show that continuous-time trajectories can be represented by a small number of




(a) Trajectory optimization

(b) Policy optimization

Figure 1.1: The tradeoffs in using dynamics models that appear in incremental trajectory optimization and episodic policy optimization. (a) In incremental trajectory optimization, full model is often too computationally expensive. The objective is to develop methods that rely on approximate models in order to meet the real-time requirement while having a small performance loss. (b) In episodic policy optimization, using an imperfect model alone without data suffers from performance bias. The goal is to develop theories and algorithms that can leverage imperfect models to accelerate learning while avoiding performance bias.

states using sparse Gaussian process (GP) models, and develop efficient incremental algorithms that combine GP representations of trajectories with structure-exploiting inference. We have adapted this idea to tackling other continuous-time trajectory optimization problems in robotics, including motion planning [3, 4] and simultaneous planning and control [5].

- (Chapter 3) We develop scalable model predictive control algorithms that utilize Gaussian process dynamics models. Our approach is based on two novel analytic moment-based closed-form expressions for Sparse Spectrum Gaussian Processes (SSGPs) prediction with uncertain inputs [6]. Unlike existing SSGP algorithms for prediction that assume deterministic inputs, our approaches are capable of reasoning about input uncertainty which is crucial in many real-world robotics and engineering applications. Compared to GP counterparts, our methods are more scalable, and can be generalized to any continuous shift-invariant kernels. Furthermore, besides the extensive experiments in simulation (Section 3.5), we verified our SSGP-based MPC algorithm on a real-world autonomous high-speed off-road driving task: the algorithm is used to realize an algorithmic expert for online imitation learning [7].

- (Chapter 4) We develop a predictor-corrector meta-algorithm, called PICCOLO, that can transform any online learning algorithm from the mirror descent family and the Follow-the-Leader family into a new hybrid method that leverages predictive models [8, 9]. Unlike previous model-based policy optimization methods, PICCOLO corrects for the mistakes of using imperfect predictive information and hence does not suffer from model bias. We show, in both theory and experiments, that the convergence rate of online policy optimization (OPO) algorithms can be improved.

- (Chapter 5) We provide new theoretical foundations of the OPO paradigm which therefore further justify the effectiveness of our predictable online learning framework PICCOLO towards solving policy optimization: we provide an explanation of the phenomenon observed in practice that the speed of policy improvement is usually much faster than existing theory suggests [10]. This new theoretical insight is corroborated by an online imitation learning experiment. We envision that the insight can provide a promising starting point to better understanding the behaviors of OPO, and to suggest directions for designing new OPO algorithms to achieve faster learning.

- (Chapter 6) We develop a class of trajectory-wise control variates (CVs) for reducing the variance of policy gradient estimate [11]. Unlike previous CVs that overlooked the variance in policy gradient estimate due to long-term trajectories, trajectory-wise CVs cancel the long-term variance by recursively augmenting existing CVs with extra terms. Like previous CVs, constructing trajectory-wise CVs requires only learning state-action value functions that can benefit from (imprecise) dynamics models. We further prove that trajectory-wise CVs are optimal for variance reduction under reasonable assumptions.

# Part I

# Incremental Trajectory Optimization

# CHAPTER 2
# INCREMENTAL TRAJECTORY ESTIMATION AND MAPPING

Recent research on simultaneous trajectory estimation and mapping (STEAM) for mobile robots started to adopt linear time-varying stochastic differential equations to model the dynamics of robots. These continuous-time linear dynamics models generate a continuous-time Gaussian process (GP) representation of robot's trajectory through its environment. Like discrete-time trajectory representations, this representation can embed prior knowledge of the robotic system in the dynamics models. But unlike discrete-time trajectory representations that have states evenly spaced in time, this continuous-time representation can elegantly handle asynchronous and sparse measurements, and allow the robot to query the trajectory to recover its estimated position at any time of interest.

A major drawback of the GP approach to STEAM rooted in the continuous-time linear dynamics is that it is formulated as a *batch* trajectory estimation problem. In this chapter we provide the critical extensions necessary to transform the existing GP-based batch algorithm for STEAM into an extremely efficient incremental algorithm. In particular, we are able to vastly speed up the solution time through incremental sparse updates and efficient variable reordering, which we believe will greatly increase the practicality of stochastic continuous-time linear dynamics models for robot mapping and localization. Finally, we demonstrate the approach and its advantages on both synthetic and real datasets. An efficient implementation of the approach developed here has been released as open source code.[1]

## 2.1 Introduction

Simultaneously recovering the location of a robot and a map of its environment from sensor readings is a fundamental challenge in robotics [12, 13, 14]. Well-known approaches to

[1]The code can be found at `https://github.com/XinyanGT/online-gpslam-code`.

this problem, such as square root smoothing and mapping (SAM) [15], have focused on regression-based methods that exploit the sparse structure of the problem to efficiently compute a solution. The main weakness of the original SAM algorithm was that it was a *batch* method: all of the data must be collected before a solution can be found. For a robot traversing an environment, the inability to update an estimate of its trajectory online is a significant drawback. In response to this weakness, Kaess et al. [16] developed a critical extension to the batch SAM algorithm, iSAM, that overcomes this problem by *incrementally* computing a solution. The main drawback of iSAM was that the approach required costly periodic batch steps for variable reordering to maintain sparsity and relinearization. This approach was extended in iSAM 2.0 [17], which employs an efficient data structure called the *Bayes tree* [18] to perform incremental variable reordering and just-in-time relinearization, thereby eliminating the bottleneck caused by batch variable reordering and relinearization. The iSAM 2.0 algorithm and its extensions are widely considered to be state-of-the-art in robot trajectory estimation and mapping.

The majority of previous approaches to to trajectory estimation and mapping, including the smoothing-based SAM family of algorithms, have formulated the problem in discrete time [19, 12, 13, 14, 15, 17, 20]. However, discrete-time representations are restrictive: they are not easily extended to trajectories with irregularly spaced waypoints or asynchronously sampled measurements. A continuous-time formulation of the SAM problem where measurements constrain the trajectory at any point in time, would elegantly contend with these difficulties. Viewed from this perspective, the robot trajectory is a *function* $\boldsymbol{x}(t)$, that maps any time $t$ to a robot state. The problem of estimating this function along with landmark locations has been dubbed *simultaneous trajectory estimation and mapping* (STEAM) [21].

Tong et al. [22] proposed a GP approach to solving the STEAM problem; the distribution over trajectories is represented by a GP that's generated from a stochastic continuous-time linear dynamics model. While their GP approach was able to accurately model and interpolate asynchronous data to recover a trajectory and landmark estimate, it suffered

from significant computational challenges: naive GP regression approaches have notoriously high space and time complexity. Additionally, Tong et al.'s approach is a *batch* method, so updating the solution necessitates saving all of the data and completely resolving the problem. In order to combat the computational burden, Tong et al.'s approach was extended by Barfoot et al. [21] to take advantage of the sparse structure inherent in the STEAM problem. Although the resulting algorithm significantly speeds up solution time, it remains a batch algorithm, which is a disadvantage for robots that need to continually update the estimate of their trajectory and environment.

In this work, we provide the critical extensions necessary to transform the existing GP-based approach to solving the STEAM problem into an extremely efficient incremental approach. Our algorithm elegantly combines the benefits of GPs and iSAM 2.0. Like the GP-based approaches to STEAM, our approach can model continuous trajectories, handle asynchronous measurements, and naturally interpolate states to speed up computation and reduce storage requirements, and, like iSAM 2.0, our approach uses a Bayes tree to efficiently calculate a *maximum a posteriori* (MAP) estimate of the GP trajectory while performing incremental factorization, variable reordering, and just-in-time relinearization. The result is an scalable incremental GP-based solution to the STEAM problem that can model continuous trajectories, handle asynchronous measurements, and naturally interpolate states to speed up computation and reduce storage requirements.

## 2.2 Batch GP-based Trajectory Estimation and Mapping

We begin by describing how the simultaneous trajectory estimation and mapping (STEAM) problem can be formulated in terms of Gaussian process regression.

### 2.2.1 GP Representation of Trajectories

Stochastic continuous-time dynamics models impose continuous-time prior distributions over state trajectories. In particular, when the dynamics is modeled by linear time-varying

(LTV) stochastic differential equations (SDE), the prior distribution becomes a GP [23, 24].

Formally, let $\boldsymbol{x}$ denote a vector-valued function of time $t$ that represent a continuous-time trajectory of a mobile robot through state-space. The LTV-SDE that models the dynamics of the robot can be written as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{v}(t) + \boldsymbol{F}(t)\boldsymbol{w}(t),\ \boldsymbol{x}(t_0) \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\mathcal{K}}_0) \tag{2.1}$$

where $\boldsymbol{v}$ is the known system control input, $\boldsymbol{A}$ and $\boldsymbol{F}$ are time-varying matrices of the system, and $\boldsymbol{w}$ is a zero-mean GP: $\boldsymbol{w} \sim \mathcal{GP}(\boldsymbol{0}, \boldsymbol{Q}_C\delta(t-t'))$ where $\boldsymbol{Q}_C$ is the power-spectral density matrix and $\delta$ is the Dirac delta function. Let $\boldsymbol{\Phi}(t,s)$ denote the state transition matrix for the continuous-time linear dynamics $\dot{\boldsymbol{x}}(t) = \boldsymbol{A}(t)\boldsymbol{x}(t)$, which transfers state from time $s$ to time $t$. Then the solution to the initial value problem of the LTV-SDE in (2.1) is

$$\boldsymbol{x}(t) = \boldsymbol{\Phi}(t,t_0)\boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t,s)(\boldsymbol{v}(s) + \boldsymbol{F}(s)\boldsymbol{w}(s))\,\mathrm{d}s. \tag{2.2}$$

It can be shown that the solution (2.2) is equivalent a GP [23, 24], i.e., $\boldsymbol{x} \sim \mathcal{GP}(\boldsymbol{\mu}, \boldsymbol{\mathcal{K}})$. The mean and kernel functions, $\boldsymbol{\mu}$ and $\boldsymbol{\mathcal{K}}$, can be calculated by taking the first and second moments of the solution (2.2).

$$\boldsymbol{\mu}(t) = \boldsymbol{\Phi}(t,t_0)\boldsymbol{\mu}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t,s)\boldsymbol{v}(s)\,\mathrm{d}s, \tag{2.3}$$

$$\boldsymbol{\mathcal{K}}(t,t') = \boldsymbol{\Phi}(t,t_0)\boldsymbol{\mathcal{K}}_0\boldsymbol{\Phi}(t',t_0)^\top + \int_{t_0}^{\min(t,t')} \boldsymbol{\Phi}(t,s)\boldsymbol{F}(s)\boldsymbol{Q}_C\boldsymbol{F}(s)^\top\boldsymbol{\Phi}(t',s)^\top\,\mathrm{d}s \tag{2.4}$$

where $\boldsymbol{\mu}_0$ and $\boldsymbol{\mathcal{K}}_0$ are the initial mean and covariance of the start state $\boldsymbol{x}(t_0)$, respectively, defined in (2.1).

### 2.2.2 Landmarks and Measurements Models

We assume that the map in the STEAM problem is represented by a set of $L$ landmarks, and the locations of the landmarks are assumed to be have a joint Gaussian distribution:

$$\boldsymbol{\ell} \sim \mathcal{N}(\boldsymbol{d}, \boldsymbol{W}), \quad \boldsymbol{\ell} = \begin{bmatrix} \boldsymbol{\ell}_1^\top & \boldsymbol{\ell}_2^\top & \dots & \boldsymbol{\ell}_L^\top \end{bmatrix}^\top \tag{2.5}$$

This, together with the GP representation of trajectories, forms a joint prior distribution over the trajectory and the map. In order to estimate the trajectory and the landmark locations, we are given $N$ measurements that are taken at $M$ measurement times.[2] Each measurement $\boldsymbol{y}_i$ is assumed to be a linear or nonlinear vector-valued function of a set of *related variables* $\boldsymbol{\theta}_i$ plus Gaussian noise $\boldsymbol{n}_i$:

$$\boldsymbol{y}_i = \boldsymbol{h}_i(\boldsymbol{\theta}_i) + \boldsymbol{n}_i, \quad \boldsymbol{n}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_i), \quad i = 1, 2, ..., N \tag{2.6}$$

The set of related variables is defined differently for different types of measurements. For instance, the related variables defined for a range measurement are the robot state at the corresponding measurement time and the associated landmark location.

### 2.2.3 Estimation of Trajectory States and Landmark Locations

To estimate the continuous-time trajectory and the landmark locations given the measurements, we can estimate the trajectory states *at measurement times* and landmarks, and then use GP state interpolation to retrieve the *entire continuous-time trajectory*. It's worthy to note that since any point along the continuous-time trajectory can be estimated using GP state interpolation, the trajectory does not need to be discretized and robot trajectory states do not need to be evenly spaced in time, which is an advantage of the GP approach over discrete-time approaches.

[2] Note that $N$ can be greater than $M$ if multiple measurements are taken simultaneously.

Based on the definition of GPs, any finite collection of robot states has a joint Gaussian distribution [25]. Thus the states at the measurement times have a joint Gaussian distribution. Formally, let $\boldsymbol{x}$ and $\boldsymbol{\mu}$ be the states at measurement times and their mean, i.e., $\boldsymbol{x} = [\ \boldsymbol{x}_1^\top \ \ \dots \ \ \boldsymbol{x}_M^\top\ ]^\top, \boldsymbol{x}_i = \boldsymbol{x}(t_i)$ and $\boldsymbol{\mu} = [\ \boldsymbol{\mu}_1^\top \ \ \dots \ \ \boldsymbol{\mu}_M^\top\ ]^\top, \boldsymbol{\mu}_i = \boldsymbol{\mu}(t_i)$, and let $\boldsymbol{\mathcal{K}}$ be the kernel matrix for the states at measurement times, i.e., $\boldsymbol{\mathcal{K}}_{ij} = \boldsymbol{\mathcal{K}}(t_i, t_j)$, then $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\mathcal{K}})$.

Therefore, the prior distribution of the combined state $\boldsymbol{\theta} = [\ \boldsymbol{x}^\top \ \ \boldsymbol{\ell}^\top\ ]^\top$ that consists of the trajectory states at measurement times and landmarks is Gaussian: $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\eta}, \boldsymbol{\mathcal{P}}), \boldsymbol{\eta} = [\ \boldsymbol{\mu}^\top \ \ \boldsymbol{d}^\top\ ]^\top, \boldsymbol{\mathcal{P}} = \operatorname{diag}(\boldsymbol{\mathcal{K}}, \boldsymbol{W})$. To estimate the combined state given the prior and the measurements, we computed the *maximum a posteriori* (MAP) estimate via Bayes' rule:

$$\begin{aligned}
\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \ p(\boldsymbol{\theta}|\boldsymbol{y}) = \arg\max_{\boldsymbol{\theta}} \ \frac{p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta})}{p(\boldsymbol{y})} \\
&= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \left(-\log p(\boldsymbol{\theta}) - \log p(\boldsymbol{y}|\boldsymbol{\theta})\right) \\
&= \arg\min_{\boldsymbol{\theta}} \ \left(\|\boldsymbol{\theta} - \boldsymbol{\eta}\|^2_{\boldsymbol{\mathcal{P}}} + \|\boldsymbol{h}(\boldsymbol{\theta}) - \boldsymbol{y}\|^2_{\boldsymbol{R}}\right)
\end{aligned} \tag{2.7}$$

where the norms are Mahalanobis norms defined as: $\|\boldsymbol{e}\|^2_{\boldsymbol{\Sigma}} = \boldsymbol{e}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{e}$, and $\boldsymbol{h}(\boldsymbol{\theta})$ and $\boldsymbol{R}$ are the mean and covariance of the measurements collected:

$$\boldsymbol{h}(\boldsymbol{\theta}) = [\ \boldsymbol{h}_1(\boldsymbol{\theta}_1)^\top \ \ \boldsymbol{h}_2(\boldsymbol{\theta}_2)^\top \ \ \dots \ \ \boldsymbol{h}_N(\boldsymbol{\theta}_N)^\top\ ]^\top, \boldsymbol{R} = \operatorname{diag}(\boldsymbol{R}_1, \boldsymbol{R}_2, \dots, \boldsymbol{R}_N)$$

Because both the covariance matrices $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{R}$ are positive definite, the objective in (2.7) corresponds to a (nonlinear) least squares problem and can be solved via iterative methods such as Gauss-Newton and Levenberg-Marquardt; in each iteration, an optimal update is computed given a linearized problem at the current estimate. A linearization of a measurement function at current state estimate $\bar{\boldsymbol{\theta}}_i$ can be accomplished by a first-order Taylor expansion:

$$\boldsymbol{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) \approx \boldsymbol{h}_i(\bar{\boldsymbol{\theta}}_i) + \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i} \delta\boldsymbol{\theta}_i \tag{2.8}$$

Combining (2.8) with (2.7), the optimal increment $\delta\boldsymbol{\theta}^*$ at the current combined state estimate $\bar{\boldsymbol{\theta}}$ is

$$\delta\boldsymbol{\theta}^* = \underset{\delta\boldsymbol{\theta}}{\arg\min} \left( \|\bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta} - \boldsymbol{\eta}\|^2_{\boldsymbol{\mathcal{P}}} + \|\boldsymbol{h}(\bar{\boldsymbol{\theta}}) + \boldsymbol{H}\delta\boldsymbol{\theta} - \boldsymbol{y}\|^2_{\boldsymbol{R}} \right) \tag{2.9}$$

where $\boldsymbol{H}$ is the measurement Jacobian matrix:$\boldsymbol{H} = \text{diag}(\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_N)$, $\boldsymbol{H}_i = \left.\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i}\right|_{\bar{\boldsymbol{\theta}}_i}$. To solve the linear least squares problem in (2.9), we take the derivative with respect to $\delta\boldsymbol{\theta}$, and set it to zero, which gives us $\delta\boldsymbol{\theta}^*$ embedded in a set of linear equations

$$\underbrace{(\boldsymbol{\mathcal{P}}^{-1} + \boldsymbol{H}^\top \boldsymbol{R}^{-1} \boldsymbol{H})}_{\boldsymbol{\mathcal{I}}} \delta\boldsymbol{\theta}^* = \boldsymbol{\mathcal{P}}^{-1}(\boldsymbol{\eta} - \bar{\boldsymbol{\theta}}) + \boldsymbol{H}^\top \boldsymbol{R}^{-1}(\boldsymbol{y} - \bar{\boldsymbol{h}}) \tag{2.10}$$

with covariance

$$\mathbb{V}[\delta\boldsymbol{\theta}^*, \delta\boldsymbol{\theta}^*] = \boldsymbol{\mathcal{I}}^{-1} \tag{2.11}$$

The positive definite matrix $\boldsymbol{\mathcal{P}}^{-1} + \boldsymbol{H}^\top \boldsymbol{R}^{-1} \boldsymbol{H}$ is the *a posteriori* information matrix, which we label $\boldsymbol{\mathcal{I}}$. To solve this set of linear equations for $\delta\boldsymbol{\theta}^*$, we do not actually have to calculate the inverse $\boldsymbol{\mathcal{I}}^{-1}$. Instead, factorization-based methods can provide a fast, numerically stable solution. For example, $\delta\boldsymbol{\theta}^*$ can be found by first performing a Cholesky factorization $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top = \boldsymbol{\mathcal{I}}$, and then solving $\boldsymbol{\mathcal{L}}\boldsymbol{d} = \boldsymbol{b}$ and $\boldsymbol{\mathcal{L}}^\top \delta\boldsymbol{\theta}^* = \boldsymbol{d}$ by back substitution. At each iteration we perform a *batch* state estimation update $\bar{\boldsymbol{\theta}} \leftarrow \bar{\boldsymbol{\theta}} + \delta\boldsymbol{\theta}^*$ and repeat the process until convergence. If $\boldsymbol{\mathcal{I}}$ is dense, the time complexity of a Cholesky factorization and back substitution are $O(n^3)$ and $O(n^2)$ respectively, where $\boldsymbol{\mathcal{I}} \in \mathbb{R}^{n \times n}$ [26]. However, if $\boldsymbol{\mathcal{I}}$ has sparse structure, then the solution can be found much faster. For example, for a narrowly banded matrix, the computation time is $O(n)$ instead of $O(n^3)$ [26]. Fortunately, we can guarantee sparsity for the STEAM problem (see Section 2.2.5 below).

### 2.2.4 State Interpolation

An advantage of the Gaussian process representation of the robot trajectory is that any trajectory state can be interpolated from the states at the measurement times [22]. Let $\bar{\boldsymbol{x}}$ denote the MAP estimate (2.7) the states at measurement times: $\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{\boldsymbol{x}}_1^\top & \dots & \bar{\boldsymbol{x}}_M^\top \end{bmatrix}^\top$. Then the state at any time $t$ can be interpolated through

$$\bar{\boldsymbol{x}}(t) = \boldsymbol{\mu}(t) + \boldsymbol{\mathcal{K}}(t)\boldsymbol{\mathcal{K}}^{-1}(\bar{\boldsymbol{x}} - \boldsymbol{\mu}), \quad \boldsymbol{\mathcal{K}}(t) = \begin{bmatrix} \boldsymbol{\mathcal{K}}(t, t_1) & \dots & \boldsymbol{\mathcal{K}}(t, t_M) \end{bmatrix}. \tag{2.12}$$

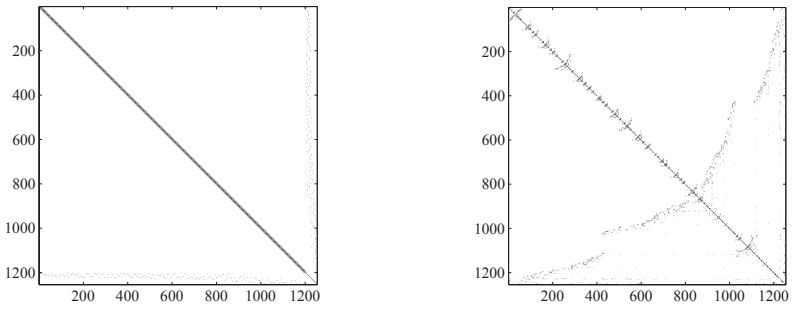
By utilizing interpolation, we can reduce the number of robot trajectory states that we need to estimate in the optimization procedure [22]. For simplicity, assume $\boldsymbol{\theta}_i$, the set of the related variables of the $i$th measurement according to the model (2.6), is $\boldsymbol{x}(\tau)$. Then, after interpolation, (2.8) becomes:

$$\begin{aligned} \boldsymbol{h}_i\left(\bar{\boldsymbol{\theta}}_i + \delta\boldsymbol{\theta}_i\right) &= \boldsymbol{h}_i\left(\bar{\boldsymbol{x}}(\tau) + \delta\boldsymbol{x}(\tau)\right) \approx \boldsymbol{h}_i(\bar{\boldsymbol{x}}(\tau)) + \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}(\tau)} \cdot \frac{\partial \boldsymbol{x}(\tau)}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}} \delta\boldsymbol{x} \\ &= \boldsymbol{h}_i\left(\boldsymbol{\mu}(\tau) + \boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}(\bar{\boldsymbol{x}} - \boldsymbol{\mu})\right) + \boldsymbol{H}_i\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}\delta\boldsymbol{x} \end{aligned} \tag{2.13}$$

By employing (2.13) during optimization, we can make use of measurement $i$ without explicitly estimating the trajectory states that it relates to. We exploit this advantage to greatly speed up the solution to the STEAM problem in practice (Section 2.5).

### 2.2.5 Sparse Gaussian Process Regression

The efficiency of the iterative methods to solving (2.10) is heavily dependent on the structure of the information matrix $\boldsymbol{\mathcal{I}}$. If the matrix is sparse, it is possible to very efficiently compute the solution [15]. For the kernel matrix $\boldsymbol{\mathcal{K}}$ of the GP generated from LTV-SDE (2.1), Barfoot et al. proved that the inverse kernel matrix $\boldsymbol{\mathcal{K}}^{-1}$ has an exact sparsity pattern: it's exactly block-tridiagonal [24]. To illustrate the sparsity pattern, let's temporarily assume that the measurements are range and odometry measurements, and the variables are ordered in XL




(a) XL ordering (b) SYMAMD ordering

Figure 2.1: Sparse information matrices. The information matrix $\boldsymbol{\mathcal{I}}$ with XL ordering (a), and SYMAMD ordering (b). Both sparse matrices have the same number of non-zero elements, yet the second matrix can be factored much more efficiently due to the heuristic ordering of the matrix columns. (See Table 2.1). For illustration, only 200 trajectory states are shown here.

ordering.[3] Then the sparse information matrix becomes $\boldsymbol{\mathcal{I}} = \begin{bmatrix} \boldsymbol{\mathcal{I}}_{xx} & \boldsymbol{\mathcal{I}}_{x\ell} \\ \boldsymbol{\mathcal{I}}_{x\ell}^\top & \boldsymbol{\mathcal{I}}_{\ell\ell} \end{bmatrix}$ where $\boldsymbol{\mathcal{I}}_{xx}$ is block-tridiagonal and $\boldsymbol{\mathcal{I}}_{\ell\ell}$ is block-diagonal. $\boldsymbol{\mathcal{I}}_{x\ell}$'s density depends on the frequency of landmark measurements, and how they are taken. See Figure 2.1a for an example.

Furthermore, when the GP is generated by LTV SDE, Equation (2.12) has a specific sparsity pattern: only two column blocks that correspond to trajectory states at $t_{i-1}$ and $t_i$ are nonzero ($t_{i-1} < \tau < t_i$) [21]:

$$\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1} = \begin{bmatrix} 0 & \dots & 0 & \boldsymbol{\Lambda}(\tau) & \boldsymbol{\Psi}(\tau) & 0 & \dots & 0 \end{bmatrix} \tag{2.14}$$

where $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$ are computed using based on the solution to LTV-SDE (2.2):

$$\boldsymbol{\Lambda}(\tau) = \boldsymbol{\Phi}(\tau, t_{i-1}) - \boldsymbol{Q}_\tau \boldsymbol{\Phi}(t_i, \tau)^\top \boldsymbol{Q}_i^{-1} \boldsymbol{\Phi}(t_i, t_{i-1})$$
$$\boldsymbol{\Psi}(\tau) = \boldsymbol{Q}_\tau \boldsymbol{\Phi}(t_i, \tau)^\top \boldsymbol{Q}_i^{-1}$$

where $\boldsymbol{\Phi}(\tau, s)$ is the state transition matrix from $s$ to $\tau$, and $\boldsymbol{Q}_\tau$ is the integral of $\boldsymbol{Q}_C$, the

[3] XL ordering is an ordering where state/process variables come before landmarks variables.

covariance of the process noise $\boldsymbol{w}(t)$ (2.1):

$$\boldsymbol{Q}_\tau = \int_{t_{i-1}}^{\tau} \boldsymbol{\Phi}(\tau, s)\boldsymbol{F}(s)\boldsymbol{Q}_C\boldsymbol{F}(s)^\top\boldsymbol{\Phi}(\tau, s)^\top ds \tag{2.15}$$

Consequently, based on (2.12) and (2.14), $\bar{\boldsymbol{x}}(\tau)$ is an affine function of only two nearby states $\bar{\boldsymbol{x}}_{i-1}$ and $\bar{\boldsymbol{x}}_i$ (the current estimate of the states at $t_{i-1}$ and $t_i$, $t_{i-1} < \tau < t_i$):

$$\bar{\boldsymbol{x}}(\tau) = \boldsymbol{\mu}(\tau) + \boldsymbol{\Lambda}(\tau)(\bar{\boldsymbol{x}}_{i-1} - \boldsymbol{\mu}_{i-1}) + \boldsymbol{\Psi}(\tau)(\bar{\boldsymbol{x}}_i - \boldsymbol{\mu}_i) \tag{2.16}$$

Thus, it only takes $O(1)$ time to query any $\bar{\boldsymbol{x}}(\tau)$ using (2.16). Moreover, because interpolation of a state is only determined by the two nearby states, measurement interpolation in (2.13) can be simplified to:

$$\begin{aligned}\boldsymbol{h}_k\left(\bar{\boldsymbol{\theta}}_k + \delta\boldsymbol{\theta}_k\right) &= \boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau) + \delta\boldsymbol{x}(\tau)\right) \approx \boldsymbol{h}_k(\bar{\boldsymbol{x}}(\tau)) + \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{x}(\tau)} \cdot \frac{\partial \boldsymbol{x}(\tau)}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}} \delta\boldsymbol{x} \\ &= \boldsymbol{h}_k(\bar{\boldsymbol{x}}(\tau)) + \boldsymbol{H}_k\Big(\boldsymbol{\Lambda}(\tau)\delta\boldsymbol{x}_{i-1} + \boldsymbol{\Psi}(\tau)\delta\boldsymbol{x}_i\Big)\end{aligned} \tag{2.17}$$

### 2.3 Batch GP-Regression with Variable Reordering

Previous work on batch continuous-time trajectory estimation as sparse Gaussian process regression [22, 21] assumes that the information matrix $\boldsymbol{\mathcal{I}}$ is sparse and applies standard block elimination to factor and solve (2.10). Despite the sparsity of $\boldsymbol{\mathcal{I}}$, for large numbers of landmarks this process can be very inefficient. Inspired by square root SAM [15], which uses variable reordering for efficient Cholesky factorization in a discrete-time context, we show that factorization-time can be dramatically improved by matrix column reordering in the sparse Gaussian process context as well.

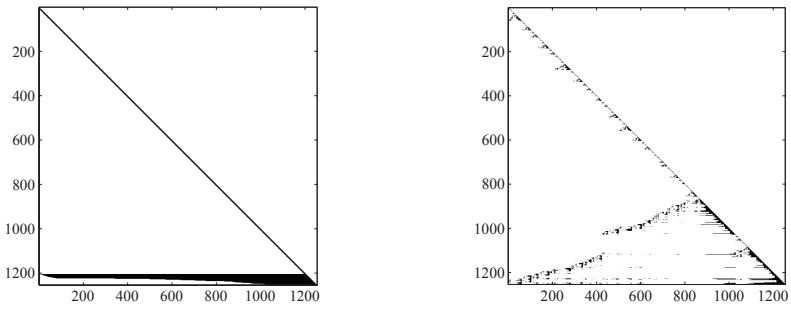
If the Cholesky decompositions are performed naively, fill-in can occur, where entries that are zero in the information matrix become non-zero in the Cholesky factor. This occurs because the Cholesky factor of a sparse matrix is guaranteed to be sparse for some variable

orderings, but not all variable orderings [27]. Therefore, we want to find a good variable ordering so that the Cholesky factor is sparse. Although finding the optimal ordering for a symmetric positive definite matrix is NP-complete [28], good heuristics do exist. One such heuristic is Symmetric Approximate Minimum Degree Permutation (SYMAMD), which is a variant of Column Approximate Minimum Degree Ordering (COLAMD) [29] on a positive definite matrix [29].

To demonstrate the benefits of variable reordering, we constructed a synthetic example and compared different approaches. The example, which is explained in detail in Section 2.5.1, consists of 1,500 time steps with trajectory states, $\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{p}_i & \dot{\boldsymbol{p}}_i \end{bmatrix}^\top$, $\boldsymbol{p}_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^\top$, and with odometry and range measurements. The total number of landmarks is 298. The structure of the information matrix $\boldsymbol{\mathcal{I}}$ and Cholesky factor $\boldsymbol{\mathcal{L}}$, with and without variable reordering, are compared in Figure 2.1 and Figure 2.2. Although variable reordering does not change the sparsity of the information matrix $\boldsymbol{\mathcal{I}}$ (Figure 2.1), it dramatically increases the sparsity of the Cholesky factor $\boldsymbol{\mathcal{L}}$ (Figure 2.2). Table 2.1 demonstrates this clear benefit of reordering. The Cholesky factor after SYMAMD ordering contains 10.6% non-zeros of XL ordering, and takes 2.83% of the time, which are significant improvements in both time and space complexity. We also experimented with block SYMAMD [15], which exploits domain knowledge to group together variables belonging to a particular trajectory state $\boldsymbol{x}(t_i)$ or landmark location $\boldsymbol{\ell}_j$ before performing SYMAMD. Empirically we found that this further improves performance.

It is straightforward to incorporate variable reordering methods like SYMAMD and block SYMAMD into the batch GP-Regression algorithm from Section 2.2: given a new batch of data, directly update the sparse information matrix $\boldsymbol{\mathcal{I}}$, reorder the variables with (block) SYMAMD, and then recompute the Cholesky factor $\boldsymbol{\mathcal{L}}$ on the way to solving for $\delta\boldsymbol{\theta}$ in (2.10).

In most STEAM problems, we are interested in estimating the robot's trajectory *as it traverses the environment*. We accomplish this by repeatedly executing the batch algorithm




(a) XL ordering (b) SYMAMD ordering

Figure 2.2: The Cholesky factors $\mathcal{L}$ of $\mathcal{I}$. In (a), $\mathcal{L}$ is computed with XL ordering, which exhibits fill-in. In (b), $\mathcal{L}$ is computed with SYMAMD ordering, which is more sparse. For illustration, only 200 states are shown here.

with variable reordering. Although this approach seems like it should be very costly, with variable reordering it is actually quite efficient. Building and factoring the sparse information matrix is much faster than the linearization step required for a single iteration of the Gauss-Newton algorithm. Since the computational bottleneck is not the Cholesky decomposition, but rather the relinearization of the measurement model, we suggest only periodic Gauss-Newton iterations.

Table 2.1: Cost of Cholesky factorization with different ordering methods including ordering time.

| | **XL** | **SYMAMD** | **Block SYMAMD** |
|---|---|---|---|
| nnz [4] | 1817k (100%) | 192k (10.6%) | 176k (9.7%) |
| time (sec) | 0.9677 (100%) | 0.0274 (2.83%) | 0.0175 (1.81%) |

## 2.4 Fast Incremental Updates

Despite the efficiency of periodic batch updates, the resulting algorithm is still a batch algorithm that requires reordering and refactoring $\mathcal{I}$, and periodically relinearizing the measurement function for all of the estimated states each time new data is collected. Here we provide the extensions necessary to avoid these costly steps and turn the naive batch algorithm into an efficient, truly incremental, algorithm. The key idea is to perform just-

in-time relinearization and to efficiently *update* an existing sparse factorization instead of re-calculating one from scratch.

### 2.4.1 The Bayes Tree Data Structure

We base our approach on iSAM 2.0 proposed by Kaess et al. [17], which was designed to efficiently solve a nonlinear estimation problem in an incremental and real-time manner by directly operating on the factor graph representation of the SAM problem. The core technology behind iSAM 2.0 is the *Bayes tree* data structure which allows for incremental variable reordering and fluid relinearization [18]. The Bayes tree data structure captures the formal equivalence between the sparse QR factorization in linear algebra and the inference in graphical models, translating *abstract updates* to a matrix factorization into *intuitive edits* to a graph. Here we give a brief introduction of Bayes trees (see [18] for details), and how they help solve the sparse Gaussian process regression incrementally.

A Bayes tree is constructed from a Bayes net, which is itself constructed from a factor graph. A factor graph is a bipartite graph $G = (\boldsymbol{\theta}, \mathcal{F}, \mathcal{E})$, representing the factorization of a function as products: $f(\boldsymbol{\theta}) = \prod_i f_i(\boldsymbol{\theta}_i)$, where $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_m\}$ are *variables*, $\mathcal{F} = \{f_1, \ldots, f_n\}$ are *factors* (functions of variables), and $\mathcal{E}$ are the *edges* that connect these two types of nodes. $e_{ij} \in \mathcal{E}$ if and only if $\theta_j \in \boldsymbol{\theta}_i$ and $f_i(\cdot)$ is a function of $\boldsymbol{\theta}_i$.

In the context of localization and mapping, a factor graph encodes the complex probability estimation problem in a graphical model. It represents the *joint density* of the variables consisting of both trajectory and mapping, and factors correspond to the soft constraints imposed by the measurements and priors. If we assume that the priors are Gaussian, measurements have Gaussian noise, and measurement functions are linear or linearized, as in Section 2.2, the joint density becomes a product of Gaussian distributions:

$$f(\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2}\sum \|\boldsymbol{A}_i\boldsymbol{\theta}_i - \boldsymbol{b}_i\|_2^2\right\} = \exp\{-\frac{1}{2}\|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_2^2\} \tag{2.18}$$

Here $\boldsymbol{A}_i$ and $\boldsymbol{b}_i$ are derived from factor $f_i(\cdot)$. $\boldsymbol{A}$ is a square-root information matrix, with $\boldsymbol{\mathcal{I}} = \boldsymbol{A}^\top \boldsymbol{A}$ [15], so the QR factor $\boldsymbol{R}$ of $\boldsymbol{A}$ is equal to the transpose of the Cholesky factor $\boldsymbol{\mathcal{L}}$ of $\boldsymbol{\mathcal{I}}$. Maximizing the joint density is equivalent to the least-square problem in (2.9).

A Gaussian process generated from linear, time-varying (LTV) stochastic differential equations (SDE), as discussed in Section 2.2.1, has a block-tridiagonal inverse kernel matrix $\boldsymbol{\mathcal{K}}^{-1}$ [21]. In other words, the resulting Gaussian process prior factors only connect consecutive pairs of states. This leads to a sparse Factor graph (Figure 2.3). The Gaussian process prior $f_{\mathcal{GP}}(\cdot)$ between $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_i$ is defined as:

$$f_{\mathcal{GP}}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \propto \exp\left\{-\frac{1}{2}\left\|\boldsymbol{\Phi}(t_i, t_{i-1})\boldsymbol{x}_{i-1} + \boldsymbol{v}_i - \boldsymbol{x}_i\right\|^2_{\boldsymbol{Q}_i}\right\} \tag{2.19}$$

where $\boldsymbol{\Phi}(t_i, t_{i-1})$ is the state transition matrix, $\boldsymbol{Q}_i$ is the integral of the covariance of the process noise: the integral from $t_{i-1}$ to $t$ (2.15), and $\boldsymbol{v}_i$ is the integral of the exogenous input $\boldsymbol{v}(t)$ (2.1): $\boldsymbol{v}_i = \int_{t_{i-1}}^{t_i} \boldsymbol{\Phi}(t_i, s)\boldsymbol{v}(s)\,\mathrm{d}s$.

An illustrative sparse factor graph example including the GP factors is presented in Figure 2.3(a). Note that although the Gaussian process representation of the trajectory is continuous in time, to impose this prior knowledge only $M - 1$ factors connecting adjacent states are required, where $M$ is the total number of states [21].

The key of just-in-time relinearization and fluid variable reordering is to identify the portion of a factor graph impacted by a new or modified factor. When adding a new measurement, a prior for a new variable, or relinearizing a previous measurement, a factor graph will change accordingly. However, most modifications only have *local* effects. For example, in a pose graph optimization setting [30], due to the fact that relative pose measurements are measurements on the two most recently accessed variables, they only affect the top of the Bayes tree, leaving branches of the tree downstream untouched [17]. Therefore, the time complexity of adding each measurement factor is $O(1)$, although the effect of loop constraints is dependent on the ordering. Exploiting this observation is the

foundation for efficient incremental updates. Identifying the impacted portion is difficult to achieve directly from a factor graph, but in a Bayes tree, it can be efficiently identified as directly affected nodes and their ascendants in the tree.

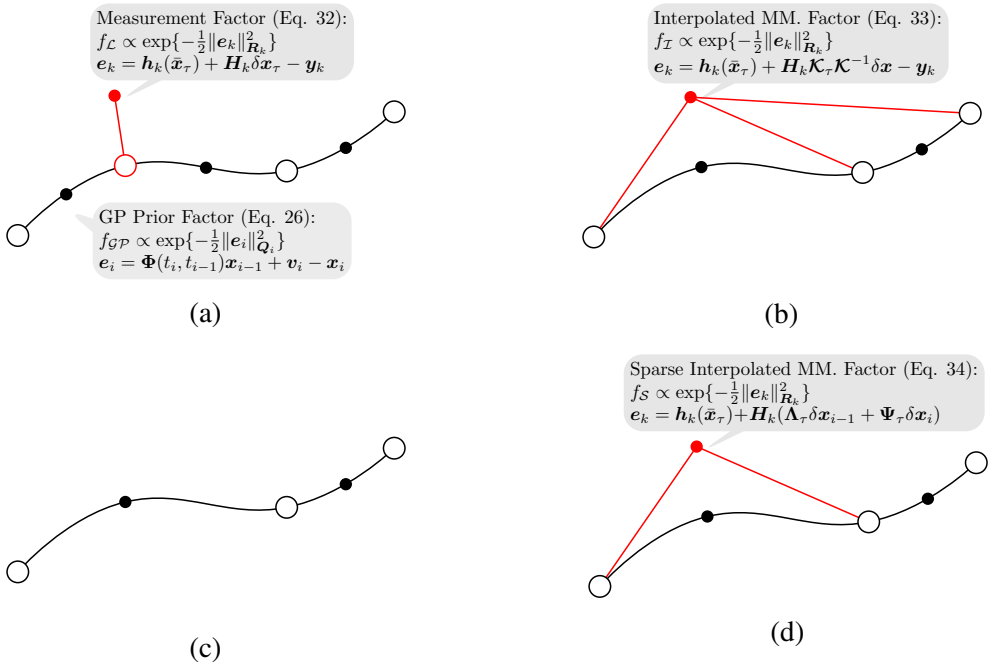



Figure 2.3: The effect of interpolation and without interpolation. (a) Measurements are fully utilized. (b) Missing state, using interpolated measurements. (c) Missing state, ignore measurements. (d) Missing state, using sparse interpolated measurement.

In summary, the Bayes tree can be used to perform fast incremental updates to the Gaussian process representation of the continuous-time trajectory. As we demonstrate in the experimental results, this can greatly increase the efficiency of Barfoot et. al's batch sparse GP algorithm when the trajectory and map need to be updated online.

Despite the interpretation of the trajectory as a Gaussian process, the approach described above is algorithmically identical to iSAM2.0 when the states associated with each measurement are explicitly estimated. In Section 2.4.2 below, we extend our incremental algorithm to use Gaussian process interpolation within the Bayes tree. By interpolating missing states, we can handle asynchronous measurements and even remove states in order to speed computation. In Section 2.5.1 and Section 2.5.2 we show that this results in a significant speedup over iSAM2.0.

### 2.4.2 Faster Updates Through Interpolation

To handle asynchronous measurements or to further reduce computation time, we take advantage of Gaussian process state interpolation, described in Section 2.2.4, within our incremental algorithm. This allows us to reduce the total number of estimated states, while still using all of the measurements, including those that involve interpolated states. By only estimating a small fraction of the states along the trajectory, we realize a large speedup relative to a naive application of the Bayes tree (see Section 2.5). This is an advantage of continuous-time GP-based methods compared to discrete-time methods like iSAM 2.0.

To use Gaussian process interpolation within our incremental algorithms, we add a new type of factors that correspond to missing states (states to be interpolated). We start by observing that, from (2.6), the *measurement* factor $f_{\mathcal{M}}(\cdot)$ derived from the measurement $h_k(\cdot)$ is:

$$f_{\mathcal{M}}(\boldsymbol{\theta}_k) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k(\bar{\boldsymbol{\theta}}_k + \delta\boldsymbol{\theta}_k) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\} \tag{2.20}$$

Without loss of generality, we assume that $\boldsymbol{x}(\tau)$ is the set of variables related to the measurement and the factor, with $t_{i-1} < \tau < t_i$, so $f^m(\cdot)$ is a unitary factor of $\boldsymbol{x}(\tau)$

$$f_{\mathcal{M}}(\boldsymbol{x}(\tau)) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau) + \delta\boldsymbol{x}(\tau)\right) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\} \tag{2.21}$$

After linearizing the factor through Eq (2.8), we arrive at:

$$f_{\mathcal{L}}(\boldsymbol{x}(\tau) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau)\right) + \boldsymbol{H}_k\delta\boldsymbol{x}(\tau) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\} \tag{2.22}$$

If $\boldsymbol{x}(\tau)$ is *missing*, then this factor can not be added to the factor graph directly, because a missing state implies that it should not be estimated explicitly. Instead of creating a new state, we interpolate the state and utilize the linearized measurement function after

*interpolation* (2.13) :

$$f_{\mathcal{I}}(\boldsymbol{x}) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k\left(\bar{\boldsymbol{x}}(\tau)\right) + \boldsymbol{H}_k\boldsymbol{\mathcal{K}}(\tau)\boldsymbol{\mathcal{K}}^{-1}\delta\boldsymbol{x} - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\} \tag{2.23}$$

We apply the interpolation equations for the *sparse* GP, (2.14) and (2.16), so that the factor becomes a function of the two nearby states (in contrast to the missing state):

$$f_{\mathcal{S}}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \propto \exp\left\{-\frac{1}{2}\|\boldsymbol{h}_k(\bar{\boldsymbol{x}}(\tau)) + \boldsymbol{H}_k\Big(\boldsymbol{\Lambda}(\tau)\delta\boldsymbol{x}_{i-1} + \boldsymbol{\Psi}(\tau)\delta\boldsymbol{x}_i\Big) - \boldsymbol{y}_k\|^2_{\boldsymbol{R}_k}\right\} \tag{2.24}$$

where $\bar{\boldsymbol{x}}(\boldsymbol{\tau})$ is specified in (2.16). The effect of interpolation and without interpolation on the factor graph is presented in Figure 2.3.

A factor graph augmented with the factors associated with measurements at missing states has several advantages: *1)* We can avoid estimating a missing state at time $t$ explicitly, but still make use of a measurement at time $t$. This allows our algorithm to naturally handle asynchronous measurements. *2)* We can also reduce the size of the Bayes tree and the associated matrices by skipping states, which results in a reduction of computation time. Empirically, we show in Section 2.5.1, and Section 2.5.2 that skipping large numbers of states can reduce computation time by almost 70% with only a small reduction in accuracy. For example, in Section 2.5.2, we show that for the Autonomous Lawnmower dataset, interpolating 4 missing states instead of directly estimating them reduces computation time by 68% with a 20% increase in an already small RMSE.

Interpolation enables extreme flexibility in how measurements can be incorporated into the trajectory estimation problem. When a new measurement arrives, it can be thrown away, it can be directly incorporated as a new state, or a factor can be added corresponding to a missing/interpolated state. Even after the decision has been made, it can be changed via edits to the Bayes tree. The incurred expense is dependent on the tree structure. Different strategies can be applied based on the current uncertainty in estimation (2.11), computing resources available, and required estimation accuracy. Research on designing strategies that

balance the trade-off for specific applications are left for future work.

The full incremental algorithm is described in Algorithm 1. In particular, when a measurement related to a missing state is received, the variables necessary to interpolate the state are identified. Since the sparse GP has a LTV SDE prior, each interpolated state is only a function of two nearby states (see (2.16)). These nearby states are therefore included into the set of variables $\boldsymbol{\theta}_{nf}$ related to the new factor (step 1). In the case that the GP relies on a different kernel matrix, the corresponding states used for interpolation can be determined from (2.12). Linearization of factors that involve missing states (step 3) is performed by incorporating state interpolation via (2.13).

**Algorithm 1** Incremental Sparse GP Regression visa the Bayes tree with Gaussian Process Priors (BTGP)

Set the sets of *affected* variables, variables involved in *new factors*, and *relinearized* variables to empty sets, $\boldsymbol{\theta}_{aff} = \boldsymbol{\theta}_{nf} = \boldsymbol{\theta}_{rl} = \varnothing$.

**while** collecting data **do**

1. Collect measurements, store as new factors. Set $\boldsymbol{\theta}_{nf}$ to the set of variables involved in the *new factors*. If $\boldsymbol{x}(\tau) \in \boldsymbol{\theta}_{nf}$ is a missing state, replace it by nearby states (2.16); If $\boldsymbol{x}(\tau) \in \boldsymbol{\theta}_{nf}$ is a new state to estimate, a GP prior (2.19) is stored, and $\boldsymbol{\theta}_{nf} = \boldsymbol{\theta}_{nf} \cup \boldsymbol{x}(\tau)$.
2. For all $\theta_i \in \boldsymbol{\theta}_{aff} = \boldsymbol{\theta}_{rl} \cup \boldsymbol{\theta}_{nf}$, remove the corresponding cliques and ascendants up to the root of the Bayes tree.
3. Relinearize the factors required to create the removed part using interpolation if missing states are involved (2.17).
4. Add the cached marginal factors from the orphaned sub-trees of the removed cliques and create a factor graph.
5. Eliminate the factor graph by a new variable ordering, create a Bayes tree, and attach back orphaned sub-trees.
6. Partially update estimate from the root to leaves, and stop when updates to variables are below a threshold.
7. Collect variables, for which the difference between the current estimate and the previous linearization point is above a threshold, into $\boldsymbol{\theta}_{rl}$.

**end while**

## 2.5 Experimental Results

We evaluate the performance of our incremental sparse GP regression algorithm for solving the STEAM problem on synthetic and real-data experiments and compare our approach

to the state-of-the-art. In particular, we evaluate how variable reordering can dramatically speed up the batch solution to the sparse GP regression problem, and how, by utilizing the Bayes tree and interpolation for incremental updates, our algorithm can yield even greater gains in the online trajectory estimation scenario. We compare:

- **PB**: Periodic batch (described in Section Section 2.2). This is the state-of-the-art algorithm presented in [21] (XL variable ordering), which is periodically executed as data is received.
- **PBVR**: Periodic batch with variable reordering (described in Section 2.3). Variable reordering is applied to achieve efficient matrix factorization.
- **BTGP**: The proposed approach - Bayes tree with Gaussian process prior factors (described in Section Section 2.4).

If the GP is only used to estimate the state at measurement times, the proposed approach offers little beyond a reinterpretation of the standard discrete-time iSAM 2.0 algorithm. Therefore, we also compare our GP-based algorithm, which leverages interpolation, to the standard Bayes tree approach used in iSAM 2.0. We show that by interpolating large fractions of the trajectory during optimization, the GP allows us to realize significant performance gains over iSAM 2.0 with minimal loss in accuracy. For these experiments we compare:

- **without interpolation**: BTGP without interpolation at a series of lower temporal resolutions. The lower the resolution, the fewer the states to be estimated. Without interpolation BTGP is algorithmically identical to iSAM 2.0 with coarse discretization of the trajectory. Measurements between two estimated states are simply ignored.
- **with interpolation**: BTGP with interpolation at a series of lower resolutions. In contrast to the above case, measurements between estimated states are fully utilized by interpolating missing states at measurement times (described in Section 2.4.2).

Table 2.2: Summary of the experimental datasets

| | # time steps | # odo. m. | # landmark m. | # landmarks | travel dist.(km) |
|---|---|---|---|---|---|
| Synthetic | 1,500 | 1,500 | 1,500 | 298 | 0.2 |
| Auto. Mower | 9,658 | 9,658 | 3,529 | 4 | 1.9 |
| Victoria Park | 6,969 | 6,969 | 3,640 | 151 | 3.5 |

- **finest estimate**: The baseline. BTGP at the finest resolution, estimating all states at measurement times. When measurements are synchronous with evenly-spaced waypoints and no interpolation is used, BTGP is identical to iSAM 2.0 applied to the full dataset with all measurements.

All algorithms are implemented with the same C++ library, GTSAM 3.2,[5] to make the comparison fair and meaningful. Evaluation is performed on three datasets summarized in Table 2.2. We first evaluate performance in a synthetic dataset (Section 2.5.1), analyzing estimation errors with respect to ground truth data. Results using real-world datasets are then presented in Section 2.5.2 and Section 2.5.3.

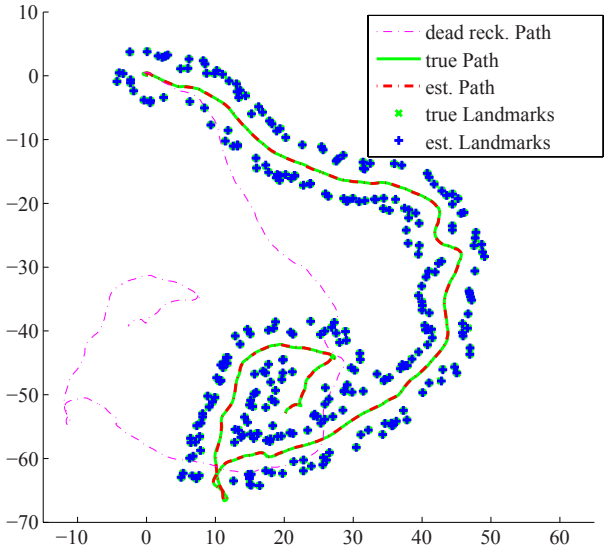



Figure 2.4: Synthetic dataset: Ground truth, dead reckoning path, and the estimates are shown. State and landmark estimates obtained from BTGP approach are very close to ground truth.

[5]https://collab.cc.gatech.edu/borg/gtsam/

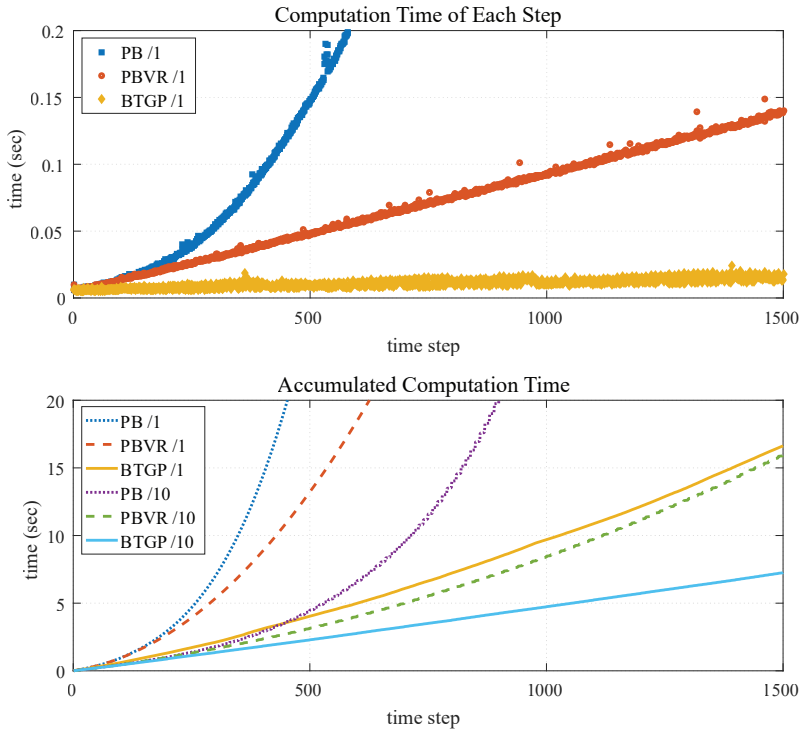



Figure 2.5: Synthetic dataset: Comparison of the computation time of three approaches PB, PBVR, and BTGP. The modifiers /1 and /10 indicate frequency of estimate updates — the number of range measurements between updates. For example: *BTGP/1* updates the estimate after 1 new range measurement using BTGP. Likewise *BTGP/10* updates the estimate after 10 new range measurements using BTGP. Due to the large number of landmarks, 298, variable reordering dramatically improves the performance.

### 2.5.1 Synthetic SLAM Exploration Task

This dataset consists of an exploration task with 1,500 time steps. Each time step contains a trajectory state $\boldsymbol{x}_i = [\ \boldsymbol{p}_i^\top \quad \dot{\boldsymbol{p}}_i^\top\ ]^\top$, $\boldsymbol{p}_i = [\ x_i \quad y_i \quad \theta_i\ ]^\top$, an odometry measurement, and a range measurement related to a nearby landmark. The total number of landmarks is 298. The trajectory is randomly sampled from a Gaussian process generated from white noise acceleration $\ddot{\boldsymbol{p}}(t) = \boldsymbol{w}(t)$, i.e.,constant velocity, and with zero mean.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{F}\boldsymbol{w}(t), \boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{p}(t) \\ \dot{\boldsymbol{p}}(t) \end{bmatrix}, \boldsymbol{p}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix}, \boldsymbol{A} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}, \boldsymbol{F} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

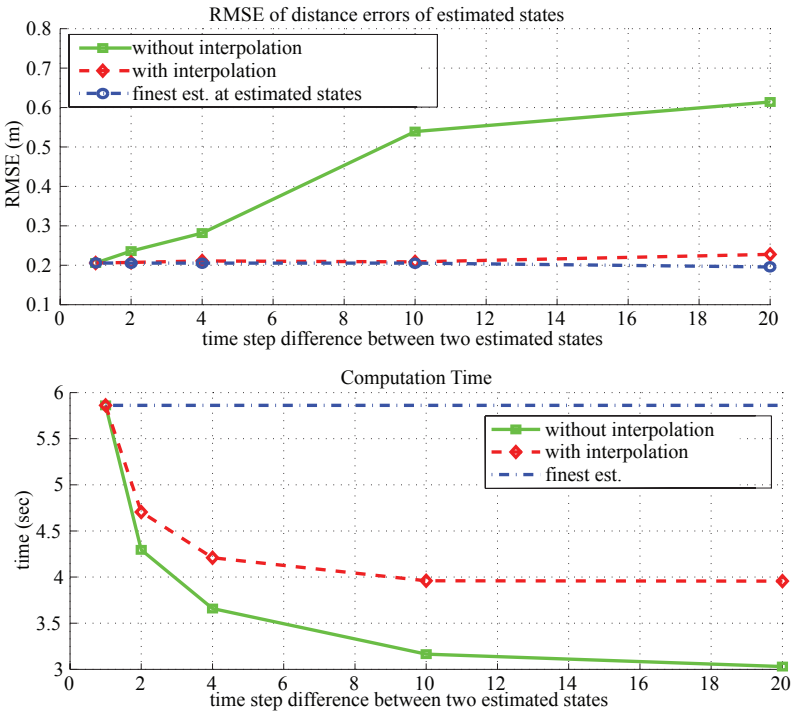



Figure 2.6: Synthetic dataset: Trade-off between computation time and accuracy if BTGP makes use of interpolation. The $y$-axis measures the RMSE of distance errors of the estimated trajectory states and total computation time with increasing amounts of interpolation. The $x$-axis measures the time step difference between two estimated (non-interpolated) states. The results indicate that interpolating $\sim 90\%$ of the states (i.e.,estimating only $\sim 10\%$ of the states) while running BTGP can result in a $33\%$ reduction in computation time over iSAM 2.0 without sacrificing accuracy.

Note that velocity $\dot{\boldsymbol{p}}(t)$ must to be included in trajectory state to represent the motion in LTV SDE form [21]. This Gaussian process representation of trajectory is also applied the other two datasets. The odometry and range measurements with Gaussian noise are specified in (2.25) and (2.26) respectively.

$$\boldsymbol{y}_{\mathcal{O}}(\boldsymbol{p}_i) = \begin{bmatrix} \dot{x}_i \cos\theta_i + \dot{y}_i \sin\theta_i \\ \dot{\theta}_i \end{bmatrix} + \boldsymbol{n}_{\mathcal{O}} \tag{2.25}$$

$$y_{\mathcal{R}}(\boldsymbol{p}_i, \boldsymbol{\ell}_j) = \left\| \begin{bmatrix} x_i & y_i \end{bmatrix}^\top - \boldsymbol{\ell}_j \right\|_2 + n_{\mathcal{R}} \tag{2.26}$$

where odometry measurements $\boldsymbol{y}_{\mathcal{O}}(\cdot)$ consists of the robot-oriented velocity and heading angle velocity with Gaussian noise, and range measurements $y_{\mathcal{R}}(\cdot)$ is the distance between the robot and a specific landmark $\boldsymbol{\ell}_j$ at $t_i$ with Gaussian noise.

We compare the computation time of the three approaches (PB, PBVR and BTGP) in Figure 2.5. The incremental Gaussian process regression (BTGP) offers significant improvements in computation time compared to the batch approaches (PBVR and PB). In Figure 2.6, we demonstrate that BTGP can further increase speed over a naive application of the Bayes tree (e.g.,iSAM 2.0) without sacrificing much accuracy by leveraging interpolation. To illustrate the trade-off between the accuracy and time efficiency due to interpolation, we plot RMSE of distance errors and the total computation time by varying the time step difference (the rate of interpolation) between estimated states. To further speed up our method while maintaining accuracy, it may be possible to dynamically specify the number of interpolated states between two estimated states, fully exploiting the flexibility provided by interpolation. This is left for future work.

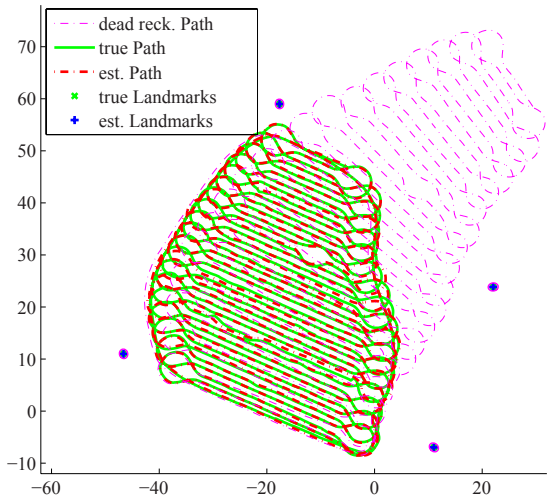



Figure 2.7: Autonomous Lawnmower dataset: Ground truth, dead reckoning path and estimates are shown. The range measurements are sparse, noisy, and asynchronous. Ground truth and the estimates of path and landmarks obtained from BTGP are very close.

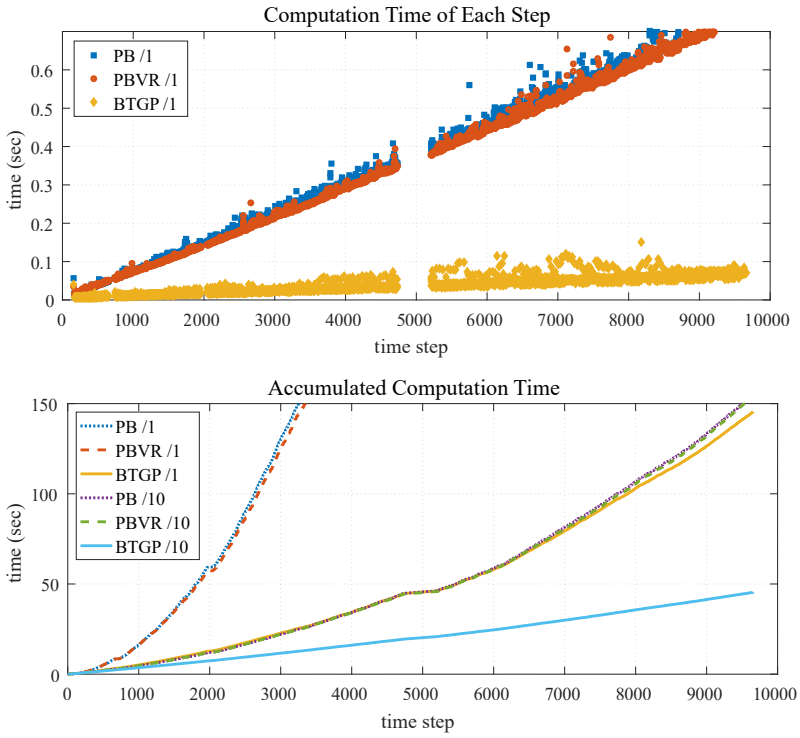



Figure 2.8: Autonomous Lawnmower dataset: Comparison of the computation time of PB, PBVR, and BTGP. As in Figure 2.5, /1 and /10 are modifiers — the number of range measurement between updates, and no interpolation is used by BTGP. The 'gap' in the upper graph is due to a long stretch around timestep 5000 with no range measurements . Due to the low number of landmarks, variable reordering does not help The incremental BTGP approach dramatically reduces computation time.

### 2.5.2 Autonomous Lawnmower

The second experiment evaluates our approach on real data from a freely available range-only SLAM dataset collected from an autonomous lawn-mowing robot [31]. The "Plaza" dataset consists of odometer data and range data to stationary landmarks collected via time-of-flight radio nodes. (Additional details on the experimental setup can be found in [31].) Ground truth paths are computed from GPS readings and have 2cm accuracy according to [31]. The environment, including the locations of the landmarks and the ground truth paths, are shown in Figure 2.7. The robot travelled 1.9km, occupied 9,658 poses, and received 3,529 range measurements, while following a typical path generated during mowing. The dataset has sparse range measurements, but contains odometry measurements at each time step. The

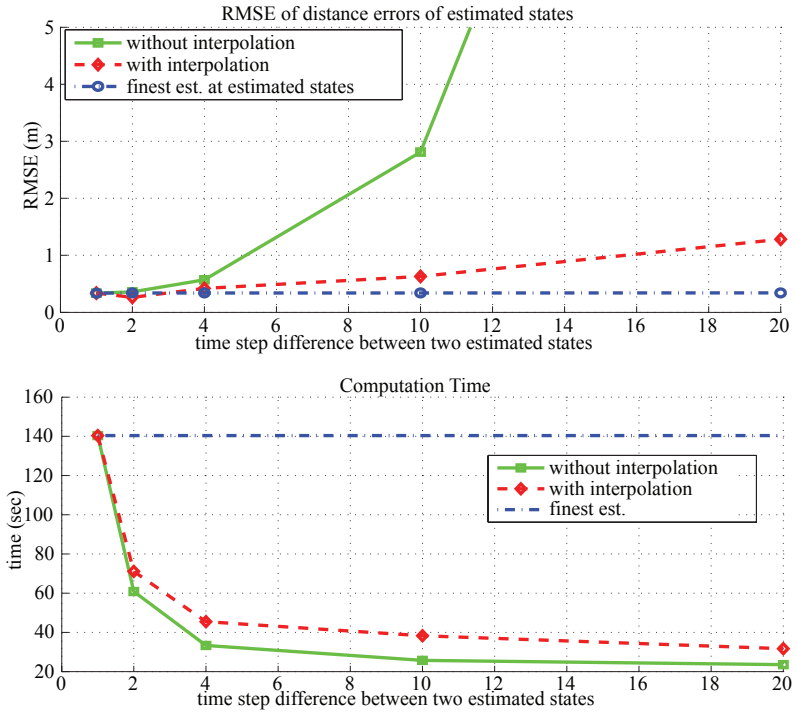



Figure 2.9: Autonomous Lawnmower dataset: Trade-off between computation time and accuracy if BTGP makes use of interpolation. The $y$-axis measures the RMSE of distance errors and total computation time with increasing amounts of interpolation. The $x$-axis measures the time step difference between two estimated (non-interpolated) states. The results indicate that interpolating $\sim 80\%$ of the states within BTGP results in only an 8cm increase in RSME while reducing the overall computation time by $68\%$ over iSAM 2.0.

results of incremental BTGP are shown in Figure 2.7 and demonstrate that we are able to estimate the robot's trajectory and map with a very high degree of accuracy.

As in Section 2.5.1, performance of three approaches – periodic batch relinearization (PB), periodic batch relinearization with variable reordering (PBVR) and incremental Bayes tree (BTGP) are compared in Figure 2.8. In this dataset, the number of landmarks is 4, which is extremely small relative to the number of trajectory states, so there is no performance gain from reordering. However, the Bayes tree-based approach dramatically outperforms the other two approaches. As the problem size increases, there is negligible increase in computation time, even for close to 10,000 trajectory states.

In Figure 2.9, the results of interpolation at different levels of resolutions are presented, which indicate a significant reduction in computation time can be achieved with minor

sacrifice in accuracy.

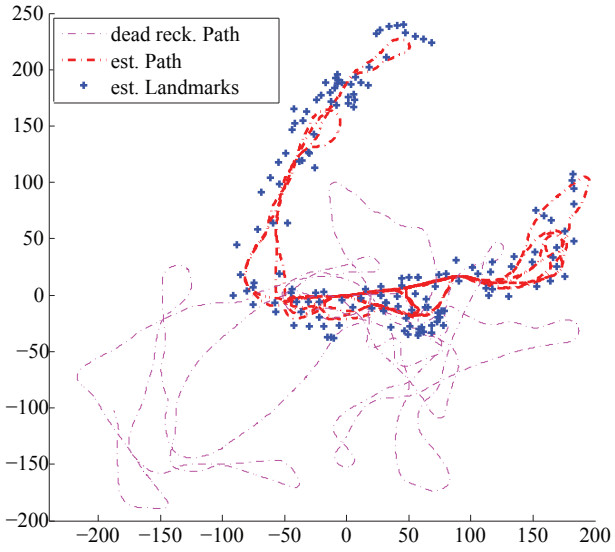



Figure 2.10: Victoria Park dataset: Dead reckoning and estimated path obtained from BTGP approach.

### 2.5.3 Victoria Park

The third experiment evaluates our approach on the Victoria Park dataset [32], which consists of range-bearing measurements to landmarks, and speed and steering odometry measurements. The data was collected from a vehicle equipped with a laser sensor driving through the Sydney's Victoria Park. The environment contains a high number of trees as landmarks. The vehicle travelled $\sim 3.5$ km in 26 minutes. After repeated measurements, taken when the vehicle is stationary, are dropped, the dataset consists of 6,969 time steps and 3,640 range-bearing measurements relative to 151 landmarks. The bearing measurement is specified in (2.27), as the relative angle from vehicle heading to the landmark direction with Gaussian noise $n_{\mathcal{B}}$:

$$y_{\mathcal{B}}(\boldsymbol{p}_i, \boldsymbol{\ell}_j) = \text{atan2}\left(y_{\ell j} - y_i,\ x_{\ell j} - x_i\right) - \theta_i + n_{\mathcal{B}} \tag{2.27}$$

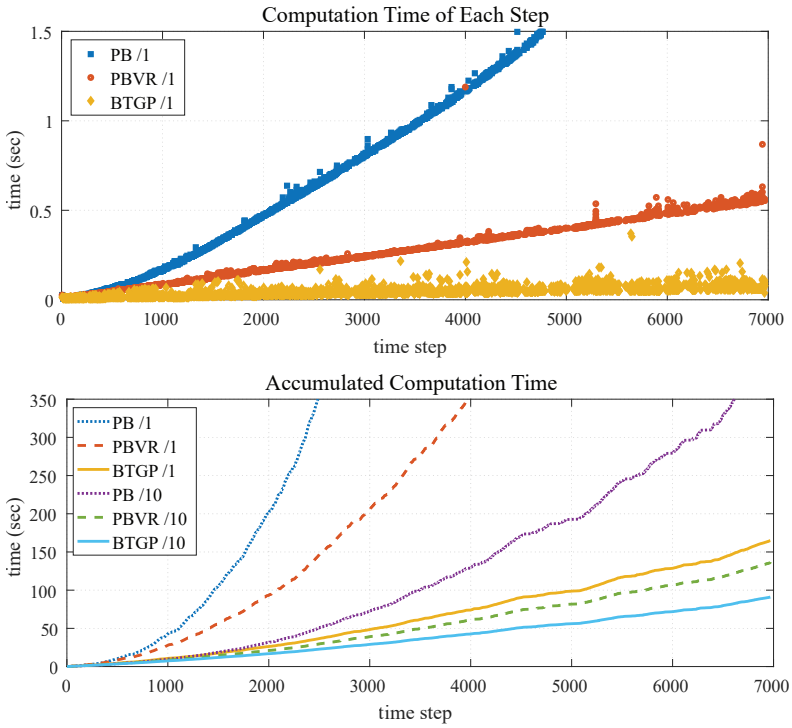



Figure 2.11: Victoria Park dataset: Comparison of the computation time of three approaches PB, PBVR, and BTGP. As in Figure 2.5 and Figure 2.8, the modifiers /1 and /10 indicate frequency of state updates. Since many landmarks are involved, PBVR dramatically improves performance, compared to PB. The incremental BTGP algorithm improves performance even further. Unlike in previous datasets, we did not evaluate the trade-off between interpolation and accuracy for Victoria Park, since we do not have access to ground truth and cannot evaluate the effect on accuracy. However, like previous datasets, interpolation can greatly increase the speed of BTGP.

where $\boldsymbol{\ell}_j = \begin{bmatrix} x_{\ell j} & y_{\ell j} \end{bmatrix}^\top$ is the location of landmark $j$, and $\boldsymbol{p}_i$ is defined the same as in Section 2.5.1. The results, shown in Figure 2.11, further demonstrate the advantages of BTGP. Variable reordering drastically reduces computation time when used within batch optimization (PBVR), and even further in the incremental algorithm (BTGP).

## 2.6 Conclusion

We have introduced an incremental sparse Gaussian process regression algorithm for computing the solution to the continuous-time simultaneous trajectory estimation and mapping (STEAM) problem. The proposed algorithm elegantly combines the benefits of Gaussian

process-based approaches to STEAM while simultaneously employing state-of-the-art innovations from incremental discrete-time algorithms for smoothing and mapping. Our empirical results show that by parameterizing trajectories with a small number of states and utilizing Gaussian process interpolation, our algorithm can realize large gains in speed over iSAM 2.0 with very little loss in accuracy (e.g.,reducing computation time by $68\%$ while increasing RMSE by only 25% (8cm) on the Autonomous Lawnmower Dataset) .

Interestingly, this idea is not limited to tackling the trajectory estimation and mapping problem; we has adapted it to solving other continuous-time trajectory optimization problems: motion planning [4] and simultaneous planning and control [5]. We viewed these two continuous-time trajectory optimization problems (over state and control sequences respectively) through the lens of probabilistic inference in akin to the trajectory estimate and mapping problem. Under this unified perspective, we proposed novel approaches that represent trajectories as sparse Gaussian processes and exploit the underlying sparsity of the problem in order to achieve fast replanning to make the robotic system robust to nonstationary environment and uncertainties in execution and sensing.

# CHAPTER 3
# PREDICTION UNDER UNCERTAINTY FOR MODEL PREDICTIVE CONTROL

## 3.1 Introduction

In Chapter 2, we looked at the problem of robot mapping and localization with a continuous-time trajectory distribution represented by a Gaussian process generated from a stochastic continuous-time linear dynamics model. In this chapter, we will focus on another widely encountered problem in robotics: model predictive control (MPC).

The objective in MPC is finding the optimal open-loop control sequence that minimizes the expected cumulative cost in *every* time step. Finding such optimal control sequence requires a dynamics model that can accurately predict a state sequence given a control sequence. In order to capture the inevitable uncertainties in the dynamics models that can arise from learning with finite amount of data and the intrinsic uncertainty in the environment, a line of research in robotics embraces Gaussian process (GP) dynamics models that can cope with uncertainty in a principled way.

Propagating uncertainty through GP dynamics model has been addressed by [33, 34], and extended to the multivariate outputs by [35]. These methods have led to the development of many algorithms in reinforcement learning [36, 37], Bayesian filtering [38, 39], and smoothing [40]. However, due to the high computational cost of GP inference and real-time optimization requirements in MPC, most GP-based control methods [37, 41, 42] are restricted to episodic reinforcement learning tasks.

A common method for approximating large-scale kernel machines is through random Fourier features [43]. The key idea is to map the input to a low-dimensional feature space yielding fast linear methods. In the context of GP regression, this idea leads to the sparse spectrum GP (SSGP) regression algorithm [44]. SSGP has been extended in a number of

ways for, e.g., incremental model learning [45], and large-scale GP regression [46, 47]. Unfortunately, existing SSGP algorithms for prediction assume deterministic inputs, precluding their use in many real-world robotics and engineering applications where accounting for input uncertainty is crucial.

We address this limitation of SSGP by proposing two analytic moment-based closed-form expressions for SSGP prediction with uncertain inputs. Our methods are more general and scalable than their standard GP counterparts, and are naturally applicable to multi-step prediction or uncertainty propagation. Using the new prediction under uncertainty methods, we present an SSGP-based MPC algorithm that is fast enough to perform probabilistic trajectory optimization and model adaptation on-the-fly. We evaluate our algorithm with comparative analyses in experiments.

### 3.2 Sparse Spectral Representation of GPs

Consider the task of learning the function $f : \mathbb{R}^d \to \mathbb{R}$, given i.i.d. data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$, with each pair related by

$$y = f(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2), \tag{3.1}$$

where $\epsilon$ is i.i.d. additive Gaussian noise. Gaussian process regression (GPR) is a principled way of performing Bayesian inference in function space, assuming that function $f$ has a prior distribution $f \sim \mathcal{GP}(m, k)$, with mean function $m : \mathbb{R}^d \to \mathbb{R}$ and kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. Without loss of generality, we assume $m(x) = 0$. Exact GPR is challenging for large datasets due to its $O(n^3)$ time and $O(n^2)$ space complexity [48], which is a direct consequence of having to store and invert an $n \times n$ kernel matrix.

Random features can be used to form an unbiased approximation of continuous shift-invariant kernel functions, and are proposed as a general mechanism to accelerate large-scale kernel machines [43], via explicitly mapping inputs to low-dimensional feature space. Based

on Bochner's theorem, the Fourier transform of a continuous shift-invariant positive definite kernel $k(x, x')$ is a proper probability distribution $p(\omega)$, assuming $k(x, x')$ is properly scaled [43]:

$$\begin{aligned} k(x, x') &= \int p(\omega) e^{j\omega^\top (x - x')} \, \mathrm{d}\omega \\ &= \mathbb{E}[\phi_\omega(x) \phi_\omega(x')^*], \quad \omega \sim p(\omega), \end{aligned} \tag{3.2}$$

where $\phi_\omega(x) = e^{j\omega^\top x}$ and the superscript star denotes complex conjugate. We can see that in (3.2) $k(x, x')$ only depends on the lag vector separating $x$ and $x'$: $x - x'$. (3.2) leads to an unbiased finite sample approximation of $k$: $k(x, x') \approx \frac{1}{m} \sum \phi_{\omega_i}(x) \phi_{\omega_i}(x')^*$, where random frequencies $\{\omega_i\}_{i=1}^m$ are drawn i.i.d. from $p(\omega)$. Utilizing the fact that $\phi_\omega$ can be replaced by sinusoidal functions since both $p(\omega)$ and $k(x, x')$ are reals, and concatenating features $\{\phi_{\omega_i}\}_{i=1}^m$ into a succinct vector form, an approximation for $k(x, x')$ is expressed as

$$k(x, x') \approx \phi(x)^\top \phi(x'), \ \phi(x) = \begin{bmatrix} \phi^c(x) \\ \phi^s(x) \end{bmatrix}, \tag{3.3}$$

$$\phi_i^c(x) = \sigma_k \cos(\omega_i^\top x), \ \phi_i^s(x) = \sigma_k \sin(\omega_i^\top x), \ \omega_i \sim p(\omega),$$

where $\sigma_k$ is a scaling coefficient. For the commonly used Squared Exponential (SE) kernel: $k(x, x') = \sigma_f^2 \exp(-\frac{1}{2} \|x - x'\|_{\Lambda^{-1}}^2)$, $p(\omega) = \mathcal{N}(0, \Lambda^{-1})$ and $\sigma_k = \frac{\sigma_f}{\sqrt{m}}$, where the coefficient $\sigma_f$ and the diagonal matrix $\Lambda$ are the hyperparameters, examples of kernels and corresponding spectral densities can be found in Table 3.1.

In accordance with this feature map (3.3), Sparse Spectrum GPs are defined as follows

**Definition 1.** Sparse Spectrum GPs (SSGPs) are GPs with kernels defined on the finite-dimensional and randomized feature map $\phi$ (3.3):

$$k(x, x') = \phi(x)^\top \phi(x') + \sigma_n^2 \delta(x - x'), \tag{3.4}$$

where the function $\delta$ is the Kronecker delta function.

The second term in (3.4) accounts for the additive zero mean Gaussian noise in (3.1), in order to directly represent the correlation between $x$ and $y$: $p(y|x)$.

Because of the explicit finite-dimensional feature map (3.3), each SSGP is equivalent to a Gaussian distribution over the weights of features $w \in \mathbb{R}^{2m}$. Assuming that prior distribution of weights $w$ is $\mathcal{N}(0, I)$ [1] and the feature map is fixed, after conditioning on the data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$, the posterior distribution of $w$ is [2]

$$\begin{aligned} w &\sim \mathcal{N}(\alpha,\ \sigma_n^2 A^{-1}), \\ \alpha &= A^{-1}\Phi Y, \quad A = \Phi\Phi^\top + \sigma_n^2 I, \end{aligned} \tag{3.5}$$

which can be derived through Bayesian linear regression. In (3.5), the column vector $Y$ and the matrix $\Phi$ are specified by the data $\mathcal{D}$: $Y = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^\top$, $\Phi = \begin{bmatrix} \phi(x_1) & \dots & \phi(x_n) \end{bmatrix}$. Consequently, the posterior distribution over the output $y$ in (3.1) at a test point $x$ is *exactly Gaussian*, in which the posterior variance explicitly captures the model uncertainty in prediction with input $x$:

$$p(y|x) = \mathcal{N}(\alpha^\top \phi(x),\ \sigma_n^2 + \sigma_n^2 \|\phi(x)\|_{A^{-1}}^2). \tag{3.6}$$

This Bayesian linear regression method for SSGP is proposed in [44]. Its time complexity is $O(nm^2 + m^3)$, which is significantly more efficient than standard GPR's $O(n^3)$ when $m \ll n$.

**Remark** It's worth noting that the methods proposed in this chapter are not tied to specific algorithms for SSGP regression such as Bayesian linear regression [44], but able to account

[1] $I$ is the identity matrix with proper size. The prior covariance is identity since $\mathbb{E}[f(x)f(x)] = \mathbb{E}\left[\phi(x)^\top w w^\top \phi(x')\right] = \phi(x)^\top \mathbb{E}[ww^\top]\phi(x')$, and $\mathbb{E}[f(x)f(x')] = \phi(x)^\top \phi(x')$ (see Section 2.2 in [36] for details.)

[2] Conditioning on data $\mathcal{D}$ is omitted, e.g., in $w|\mathcal{D}$, for simplicity in notation.

for any SSGP with specified feature weights distribution (3.5), where posterior $\alpha$ and $A$ can be computed by any means. Variations on $A$ include sparse approximations by a low rank plus diagonal matrix, or iterative solutions by optimization methods like doubly stochastic gradient descent [46].

### 3.3 Prediction under Uncertainty

The problem of *prediction under uncertainty*, appears in many fields of science and engineering that involve sequential prediction including state estimation [38, 40], time series prediction [34], stochastic process approximation [49], and planning and control [37]. In these problems, uncertainty can be found in both the predictive models and the model's inputs. Formally, we are often interested in finding the probability density of a prediction $y$, given a distribution $p(x)$ and a probabilistic model $p(y|x)$. By marginalization,

$$p(y) = \int p(y|x)p(x)\,\mathrm{d}x. \tag{3.7}$$

Unfortunately, computing this integral exactly is often intractable.

In this chapter, two methods for prediction under uncertainty are presented under two conditions: *1)* the uncertain input is normally distributed: $x \sim \mathcal{N}(\mu, \Sigma)$, and *2)* probabilistic models are in the form of (3.6) specified by SSGPs. Despite these conditions, evaluating the integral in (3.7) is still intractable. In this work, we approximate the true predictive distribution $p(y)$ by a Gaussian distribution with moments that are analytically computed through: *1)* exact moment matching, and *2)* linearization of posterior mean function. Closed-form expressions for predictive mean, variance, covariance, and input-prediction cross-covariance are derived.

We handle multivariate outputs, i.e., multi-dimensional state, by utilizing conditionally independent scalar models for each output dimension, i.e., assuming for outputs in different dimension $y_a$ and $y_b$, $p(y_a, y_b|x) = p(y_a|x)p(y_b|x)$. Discussions on this assumption can be

found in Section A.2.1. For notational simplicity, we suppress the dependency of $\phi(x)$ on $x$, and treat $y$ as a scalar by default.

### 3.3.1 Exact Moment Matching (SSGP-EMM)

We derive the closed-form expressions for exact moments: *1)* the predictive mean $\mathbb{E}[y]$, *2)* the predictive variance $\mathbb{V}[y]$ and covariance $\mathbb{V}[y_a, y_b]$, which in the multivariate case correspond to the diagonal and off-diagonal entries of the predictive covariance matrix, and 3) the cross-covariance between input and prediction $\mathbb{V}[x, y]$.

Using the expressions for SSGP (3.3), (3.6), and the law of total expectation, the predictive mean becomes

$$\mathbb{E}[y] = \mathbb{E}\mathbb{E}[y|x] = \mathbb{E}\left[\alpha^\top \phi\right] = \alpha^\top \mathbb{E}\begin{bmatrix}\phi^c \\ \phi^s\end{bmatrix}, \tag{3.8}$$

$$\mathbb{E}[\phi_i^c] = \sigma_k \mathbb{E}[\cos(\omega_i^\top x], \quad \mathbb{E}[\phi_i^s] = \sigma_k \mathbb{E}[\sin(\omega_i^\top x],$$

where $i = 1, \ldots, m$, and in the nested expectation $\mathbb{E}\mathbb{E}[y|x]$, the outer expectation is over the input distribution $p(x) = \mathcal{N}(\mu, \Sigma)$, and the inner expectation is over the conditional distribution $p(y|x)$ (3.6).

By observing (3.8), we see that the expectation of sinusoids under the Gaussian distribution is the key to computing the predictive mean. Thus, we state the following proposition:

**Proposition 1.** *The expectation of sinusoids over multivariate Gaussian distributions:* $x \sim \mathcal{N}(\mu, \Sigma)$, $x \in \mathbb{R}^d$, *i.e.,* $p(x) = (2\pi)^{-\frac{d}{2}}(\det \Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2}\|x-\mu\|^2_{\Sigma^{-1}})$, *can be computed analytically:*

$$\mathbb{E}[\cos(\omega^\top x] = \exp(-\frac{1}{2}\|\omega\|^2_\Sigma)\cos(\omega^\top \mu),$$
$$\mathbb{E}[\sin(\omega^\top x] = \exp(-\frac{1}{2}\|\omega\|^2_\Sigma)\sin(\omega^\top \mu).$$

To prove it, we invoke Euler's formula to transform the left-hand-side to complex domain, apply identities involving quadratic exponentials, and then convert back to real numbers (see Section A.1 for details). In Proposition 1, the expectations depend on the mean and variance of the input Gaussian distribution. Intuitively, after passing a Gaussian distributed input through a sinusoidal function, the expectation of the output is equal to passing the mean of the input through the sinusoid, and then scaling it by a constant $\exp(-\frac{1}{2}\|\omega\|_\Sigma^2)$, which depends on the variance of the input. Expectations are smaller with larger input variance due to the periodicity of sinusoids.

The exact moments are then derived using Proposition 1. By the law of total variance, the predictive variance is

$$\begin{aligned}\mathbb{V}[y] &= \mathbb{E}\mathbb{V}[y|x] + \mathbb{V}\mathbb{E}[y|x] \\ &= \sigma_n^2 + \sigma_n^2 \mathrm{Tr}\left(A^{-1}\Psi\right) + \alpha^\top \Psi \alpha - \mathbb{E}[y]^2,\end{aligned} \tag{3.9}$$

where $\Psi$ is defined as the expectation of the outer product of feature vectors over input distribution $p(x)$. Specifically, we compute $\Psi$ by applying the product-to-sum trigonometric identities:

$$\begin{aligned}\mathbb{E}\left[\phi\phi^\top\right] = \Psi &= \begin{bmatrix} \Psi^{cc} & \Psi^{cs} \\ \Psi^{sc} & \Psi^{ss} \end{bmatrix}, \\ \Psi_{ij}^{cc} &= \frac{\sigma_k^2}{2}\left(\mathbb{E}\left[\cos(\omega_i + \omega_j)^\top x\right] + \mathbb{E}\left[\cos(\omega_i - \omega_j)^\top x\right]\right), \\ \Psi_{ij}^{ss} &= \frac{\sigma_k^2}{2}\left(\mathbb{E}\left[\cos(\omega_i - \omega_j)^\top x\right] - \mathbb{E}\left[\cos(\omega_i + \omega_j)^\top x\right]\right), \\ \Psi_{ij}^{cs} &= \frac{\sigma_k^2}{2}\left(\mathbb{E}\left[\sin(\omega_i + \omega_j)^\top x\right] - \mathbb{E}\left[\sin(\omega_i - \omega_j)^\top x\right]\right),\end{aligned}$$

where $\Psi^{cc}, \Psi^{ss}, \Psi^{cs}$ are $m \times m$ matrices, and $i, j = 1, \ldots, m$, on whose terms Proposition 1 can be directly applied.

Next, we derive the covariance for different output dimensions for multivariate prediction. These correspond to the off-diagonal entries of the predictive covariance matrix. We

show that, despite the conditional independence assumption for different outputs given a deterministic input, outputs become coupled with uncertain inputs. Using the law of total covariance, the covariance is

$$\begin{aligned}\mathbb{V}[y_a, y_b] &= \mathbb{V}\left[\mathbb{E}[y_a|x], \mathbb{E}[y_b|x]\right] \\ &= \mathbb{E}\left[\mathbb{E}[y_a|x]\mathbb{E}[y_b|x]\right] - \mathbb{E}[y_a]\mathbb{E}[y_b] \\ &= \alpha_a^\top \Psi_{ab} \alpha_b - (\alpha_a^\top \mathbb{E}[\phi_a])(\alpha_b^\top \mathbb{E}[\phi_b]),\end{aligned} \tag{3.10}$$

where matrix $\Psi_{ab}$ is the expectation of the outer product of feature vectors corresponding to different feature maps $\phi_a$, $\phi_b$ for outputs $y_a$, $y_b$, computed similarly as in (3.3.1) with corresponding random frequencies $\{\omega_i\}$, and the scaling coefficient $\sigma_k$ (3.3). Vectors $\alpha_a$ and $\alpha_b$ are the corresponding weight vectors for $y_a$ and $y_b$ (3.6). Compared to the expression for the variance of a single output in (3.9), the term $\mathbb{E}\left[\mathbb{V}[y_a, y_b|x]\right]$ that is included in the law of total covariance is neglected due to the assumption of conditional independence of different outputs (Section 3.2), so (3.10) does not have the corresponding first two terms in (3.9).

Finally, we compute the cross-covariance between input and each output dimension. Invoking the law of total covariance:

$$\begin{aligned}\mathbb{V}[x, y] &= \mathbb{V}[x, \mathbb{E}[y|x]] \\ &= \mathbb{E}\left[x\mathbb{E}[y|x]\right] - \mathbb{E}[x]\mathbb{E}[y] \\ &= \Upsilon\alpha - \mathbb{E}[y]\mu,\end{aligned} \tag{3.11}$$

where matrix $\Upsilon$ is the expectation of the outer product of the input $x$ and the feature vector $\phi(x)$ over input distribution $x \sim \mathcal{N}(\mu, \Sigma)$:

$$\mathbb{E}[x\phi^\top] = \Upsilon = \begin{bmatrix}\Upsilon_1^c & \dots & \Upsilon_m^c & \Upsilon_1^s & \dots & \Upsilon_m^s\end{bmatrix},$$

$$\Upsilon_i^c = \sigma_k \mathbb{E}\left[\cos(\omega_i^\top x)x\right], \quad \Upsilon_i^s = \sigma_k \mathbb{E}\left[\cos(\omega_i^\top x)x\right],$$

where $i = 1, \ldots, m$. We state the following proposition to compute each column in $\Upsilon$ consisting of expectations of the product sinusoidal functions and inputs.

**Proposition 2.** *The expectation of the multiplication of sinusoids and linear functions over multivariate Gaussian distributions: $x \sim \mathcal{N}(\mu, \Sigma)$, can be computed analytically:*

$$\mathbb{E}\left[\cos(\omega^\top x)x\right] = \mathbb{E}[\cos(\omega^\top x)]\mu - \mathbb{E}[\sin(\omega^\top x)]\Sigma\omega,$$
$$\mathbb{E}\left[\sin(\omega^\top x)x\right] = \mathbb{E}[\sin(\omega^\top x)]\mu + \mathbb{E}[\cos(\omega^\top x)]\Sigma\omega,$$

*where the right-hand-side expectations have analytical expressions (Proposition 1).*

To prove it, we find an expression for $\mathbb{E}\left[a^\top x \cos(\omega^\top x)\right]$, for any $a$, through the complex domain trick used to prove Proposition 1. Next, the result is extended to $\mathbb{E}\left[x\cos(\omega^\top x)\right]$, by setting $a$ to consist of indicator vectors (see Section A.1 for details). Applying Proposition 1 and Proposition 2, we complete the derivation of $\mathbb{V}[x, y]$ in (3.11).

**Remark** In summary, SSGP-EMM computes the exact posterior moments. This is equivalent to expectation propagation [50] by minimizing the Kullback-Leibler divergence between the true distribution and its Gaussian approximation with respect to the natural parameters. SSGP-EMM's computation complexity is $O\left(m^2k^2d^2\right)$, where $m$ is the number of features, $k$ is the output dimension, and $d$ is the input dimension. The most computationally demanding part is constructing matrices $\Psi_{ab}$ (3.10) for each output pair, where each requires $O\left(m^2d^2\right)$.

Compared to the multivariate moment-matching approach for GPs (GP-EMM) [34, 35] with $O\left(n^2k^2d^2\right)$ time complexity, SSGP-EMM is more efficient when $m \ll n$. Moreover, our approach is applicable to any positive-definite continuous shift-invariant kernel with different spectral densities (see examples in Table 3.1), while previous approaches like GP-EMM [35] are only derived for squared exponential (SE) or polynomial kernels. Next we introduce a more computationally efficient but less accurate approach that avoids the computation of $\Psi_{ab}$'s.

Table 3.1: Examples of continuous shift-invariant positive-definite kernels and their corresponding spectral densities, where $r = \frac{\sqrt{2\nu}\|x-x'\|_2}{\ell}$, $K_\nu$ is a modified Bessel function, and $h = \frac{2^d \pi^{\frac{d}{2}} \Gamma(\nu+\frac{d}{2})(2\nu)^\nu}{\Gamma(\nu)\ell^{2\nu}}$.

| Kernel | $k(x, x')$ | $p(\omega)$ |
|---|---|---|
| Gaussian | $\exp(-\frac{1}{2}\|x - x'\|^2_{\Lambda^{-1}})$ | $\mathcal{N}(0, \Lambda^{-1})$ |
| Laplacian | $\exp(-\|x - x'\|_1)$ | $\prod_{i=1}^d \frac{1}{\pi(1+\omega_i)}$ |
| Matérn | $\frac{2^{1-\nu}}{\Gamma(\nu)} r^\nu K_\nu(r)$ | $h(\frac{2\nu}{\ell^2} + 4\pi^2\|\omega\|_2^2)^{\nu+\frac{d}{2}}$ |

### 3.3.2 Linearization (SSGP-Lin)

An alternative approach to computing the exact moments of the predictive distribution is based on the linearization of the posterior mean function in (3.6) at the input mean $\mu$:

$$m(x) = \alpha^\top \phi(x) \approx m(\mu) + \alpha^\top \underbrace{D\phi(\mu)}_{M}(x - \mu), \tag{3.12}$$

where $D\phi(\mu)$ denotes taking the derivative of function $\phi$ at $\mu$. Given the definition of $\phi$ in (3.3), $D\phi$ can be found by chain rule: $D\phi_i^c(x) = -\sigma_k \sin(\omega_i^\top x)\omega_i^\top$, $D\phi_i^s(x) = \sigma_k \cos(\omega_i^\top x)\omega_i^\top$.

Utilizing the linearized posterior mean function (3.12), the predictive moments can be approximated. The predictive mean approximation is

$$\mathbb{E}[y] = \mathbb{E}\mathbb{E}[y|x] \approx m(\mu), \tag{3.13}$$

and the predictive variance approximation is

$$\begin{aligned} \mathbb{V}[y] &= \mathbb{E}\mathbb{V}[y|x] + \mathbb{V}\mathbb{E}[y|x] \\ &\approx \mathbb{V}[y|\mu] + \mathbb{V}[\alpha^\top M x] \\ &= \sigma_n^2 + \sigma_n^2 \|\phi(\mu)\|^2_{A^{-1}} + \alpha^\top M \Sigma M^\top \alpha. \end{aligned} \tag{3.14}$$

and the approximate covariance between output dimension $a$ and $b$ is

$$\begin{aligned}\mathbb{V}[y_a, y_b] &= \mathbb{V}\left[\mathbb{E}[y_a|x], \mathbb{E}[y_b|x]\right] \\ &= \mathbb{E}\left[\alpha_a^\top M_a (x-\mu)(x-\mu)^\top M_b^\top \alpha_b\right] \\ &\approx \alpha_a^\top M_a \Sigma M_b^\top \alpha_b,\end{aligned} \tag{3.15}$$

where $M_a$ and $M_b$ are defined as $M$ in (3.12), except that they correspond to feature maps $\phi_a$ and $\phi_b$. Notice that the assumption of conditional independence between different outputs is invoked here again, cf., (3.10).

Finally, the cross-covariance between the input and output can be approximated as

$$\begin{aligned}\mathbb{V}[x, y] &= \mathbb{V}[x, \mathbb{E}[y|x]] \\ &\approx \mathbb{E}\left[(x-\mu)(\alpha^\top M(x-\mu))\right] \\ &= \alpha^\top M \Sigma\end{aligned} \tag{3.16}$$

Unlike SSGP-EMM, which computes exact moments (Section 3.3.1), this linearization-based approach SSGP-Lin computes an *approximation* of the predictive moments. In contrast to SSGP-EMM's $O\big(m^2k^2d\big)$ computational complexity, the computation time of SSGP-Lin is reduced to $O\big(m^2kd\big)$, as a direct consequence of avoiding the construction of $\Psi$ (3.3.1) in SSGP-EMM (3.10), which makes SSGP-Lin more efficient than SSGP-EMM, especially when the output dimension is high.

Both SSGP-EMM and SSGP-Lin are applicable to a general family of kernels. See Table 3.2 for a comparison between our methods and GP-EMM [34, 35]. In the next section, we compare these approaches in applications of filtering and control.

## 3.4 Model Predictive Control under Uncertainty

Next, we apply the proposed methods to solving a model predictive control (MPC) problem.

Table 3.2: Comparison of our proposed methods and GP-EMM [34, 35] in terms of computational complexity and generalizability.

| Method | SSGP-EMM | SSGP-Lin | GP-EMM |
|---|---|---|---|
| Time | $O(m^2k^2d^2)$ | $O(m^2k + mk^2d)$ | $O(n^2k^2d^2)$ |
| Applicable kernels | continuous shift-invariant kernels | continuous shift-invariant kernels | SE or polynomial kernels |

### 3.4.1 Problem Setting

We consider Gauss-Markov dynamics, expressed as

$$x_{t+1} = x_t + f(x_t, u_t) + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Sigma_\epsilon), \tag{3.17}$$

where subscript $t$ denotes discrete time step, $x_t$ is the state, $u_t$ is the control, and $\epsilon_t$ is the i.i.d. process noise. We call the deterministic function $f$ the *dynamics function*. We consider the scenarios where the dynamics function $f$ and the variance of the noise $\Sigma_\epsilon$ are *unknown* but a dataset $\mathcal{D} = \{(x_k, u_k, x_{k+1})\}_{k=1}^n$ is provided.

After learning $f$ and $\Sigma_\epsilon$ in (3.17) using the dataset, the goal of MPC is to find finite-horizon optimal control at *each time step*. Formally, at time step $t$, given the distribution of state $x_t$, i.e., $p(x_t)$, MPC controller finds an open-loop control sequence that minimizes the expected cumulative cost:

$$u^\star_{t:t+T-1} = \underset{u_{t:t+T-1}}{\arg\min}\, \mathbb{E}\big[h(x_{t+T}) + \sum_{i=0}^{T-1} l(x_{t+i}, u_{t+i})\big],$$

subject to the stochastic system dynamics (3.17), where functions $h$ and $l$ are the final and running cost respectively, and $T$ is the horizon.

### 3.4.2 MPC via Probabilistic Trajectory Optimization

There are two main practical challenges to solving the MPC problem above: *1)* it requires taking into account the uncertainty in the dynamics (3.17) that's due to the generalization

error in learning from the finite dataset $\mathcal{D}$ and the intrinsic process noise $\epsilon_t$, and *2)* online optimization is very computationally expensive. We address these two challenges by a combination of *1)* trajectory optimization in the belief space, and *2)* fast prediction under uncertainty.

In order to explicitly incorporate the uncertainty in the dynamics (3.17), we perform trajectory optimization in the *Gaussian belief space*. To accomplish this, we first lift the original state to Gaussian belief state, which leads to a regular optimization problem without stochasticity, and then apply differential dynamic programming (DDP) that's a common trajectory optimization algorithm designed for non-stochastic problems to find the optimal control control sequence [51, 52]. Formally, let $b_t = [\mu_t^\top \text{ vec}(\Sigma_t)^\top]^\top$ be the Gaussian belief state, where $\text{vec}(\Sigma_t)$ stands for the vectorization of $\Sigma_t$. In words, the Gaussian belief state $b_t$ is a Gaussian approximation of the distribution of the original state at time steps $t$. Then using belief states, the optimization problem in (3.18) can be approximated by

$$u^\star_{t:t+T-1} = \underset{u_{t:t+T-1}}{\arg\min}\, H(b_{t+T}) + \sum_{i=0}^{T-1} L(b_{t+i}, u_{t+i}) \tag{3.18}$$

subject to belief space dynamics (cf. (3.17)) :

$$\mu_{t+1} = \mu_t + \mathbb{E}[f_t],\ \Sigma_{t+1} = \Sigma_t + \mathbb{V}[f_t] + \mathbb{V}[x_t, f_t] + \mathbb{V}[f_t, x_t]. \tag{3.19}$$

which can be written compactly as

$$b_{t+1} = \mathcal{F}(b_t, u_t), \tag{3.20}$$

The function $H$ and $L$ in (3.18) are defined based on the second-order approximation of $h$ and $l$ around a nominal state and control sequence so that $L(b_t, u_t) \approx \mathbb{E}[l(x_t, u_t)], x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ and $H(b_T) \approx \mathbb{E}[h(x_T)], x_T \sim \mathcal{N}(\mu_T, \Sigma_T)$.

### 3.4.3 Gaussian Belief Space Dynamics using SSGP

The computation bottleneck of applying DDP in the resultant Gaussian belief state is in computing the Gaussian belief space dynamics (3.20). To achieve fast belief state propagation, we use SSGP regression to learn the dynamics function $f$ in (3.17) from the dataset $\mathcal{D} = \{(x_t, u_t, x_{t+1})\}_{t=1}^{n}$ and use SSGP-EMM or SSGP-Lin to efficiently obtain approximate Gaussian distribution over trajectory of states. Importantly, to optimize over the control sequence, DDP requires the computation of first-order derivative of the dynamics respect to both control and state. Our analytic moment expressions provide a robust and efficient way to compute these derivatives.[3]

The use of SSGP also enables us to update the dynamics model online. After more data are collected during execution, within the SSGP framework, we may incrementally *update* the posterior distribution over the feature weights $w$ (3.5) without storing or inverting the matrix $A$ explicitly. Instead we keep track of its upper triangular Cholesky factor $A = R^\top R$ [45]. Given a new sample, a rank-1 update is applied to the Cholesky factor $R$, which requires $O(m^2)$ time. To cope with time-varying systems and to make the method more adaptive, we employ a forgetting factor $\lambda \in (0, 1)$, such that the impact of the previous samples decays exponentially in time [55].

### 3.4.4 Comparison with Other Methods

Our proposed MPC algorithm, summarized in Algorithm 2, is related to several algorithms and differs in both model and controller learning (listed in Table 3.3). First, SSGPs are more robust to modeling error than Locally Weighted Projection Regression (LWPR) used in iLQG-LD [56]. See a numerical comparison in [45]. Second, we efficiently propagate uncertainty in multi-step prediction which is crucial in MPC. In contrast, AGP-iLQR [57] drops the input uncertainty and uses subset of regressors (SoR-GP) which lacks a principled

[3] In the experiments, to deal with multivariate outputs, the assumption of conditional independence between any two output dimensions is imposed. To accommodate conditional dependence in a principled way, vector-valued Gaussian processes can be used [53, 54].

way to select reference points. In addition, PDDP [41] uses GPs which are computationally expensive for online optimization. Two deep neural networks are used for modeling in [58], which make it difficult to perform online incremental learning, as we do here.

Table 3.3: Comparison of trajectory optimization-related methods with learned dynamics models. They include: our proposed algorithm, iLQG-LD [56], PDDP [41], AGP-iLQR [57], Minimax DDP [59] and SDDP with NN [58].

| | Our method | iLQG-LD | PDDP | AGP-iLQR | Minimax DDP | SDDP with NN |
|---|---|---|---|---|---|---|
| Uncertainty propagation | Yes | No | Yes | No | No | Yes |
| Dynamics model | SSGP | LWPR | GP | SoR-GP | RFWR (partial) | Neural Network |
| Online model/policy update | Yes/Yes | Yes/No | No/No | Yes/Yes | No/No | No/No |

**Algorithm 2** MPC via probabilistic trajectory optimization (1-3: offline optimization, 4-8: online optimization)

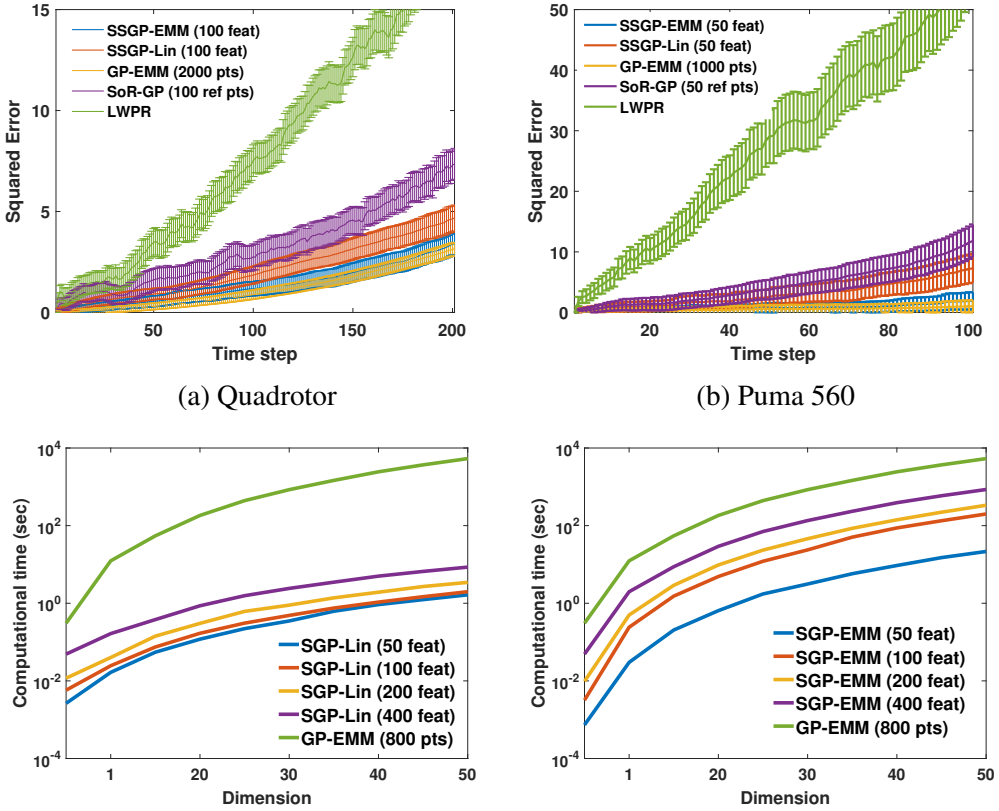
1: Model learning: collect dataset $\mathcal{D}$, and learn SSGP dynamics model (Section 3.2).
2: Initialization: set $t = 0$, and estimate $p(x_0)$.
3: Trajectory optimization: perform trajectory optimization in belief space, obtain $u^{\star}_{t:t+T-1}$.
4: **repeat**
5: Policy execution: apply one-step control $u^{\star}_{t+1}$ to the system and move one step forward, update $t = t + 1$.
6: Model adaptation: incorporate new data and update SSGP dynamics model.
7: Trajectory optimization: perform re-optimization with the updated model. Initialize with the previously optimized trajectory and obtain new $u^{\star}_{t:t+T-1}$.
8: **until** Task terminated

### 3.5 Experimental Results

In this section, we evaluate our algorithms with comparative analyses in experiments. The experiments consist of three parts: *1)* We investigate the computational demand for one-step prediction using SSGP-EMM, SSGP-Lin, and full GP-EMM [33, 34, 35]; *2)* We compare the accuracy in multi-step prediction between SSGP-EMM, SSGP-Lin and three existing

approaches: GP-EMM, Subset of Regressors GP (SoR-GP) [48] used in AGP-iLQR [57], and LWPR [60] used in iLQG-LD [56]; *3)* We study how the accuracy in multi-step prediction influences the performance on the performance on MPC tasks by comparing the performance between Algorithm 2 that's based on SSGP-EMM and SSGP-Lin and existing methods: PDDP [41] that's based on GP-EMM, AGP-iLQR, and iLQG-LD.

The cost reduction results averaged over 3 independent trials are shown.




(a) Quadrotor

(b) Puma 560

(c) Computation time

(d) Computation time

Figure 3.1: (a)-(b): Approximate inference accuracy test. The vertical axis is the squared error of cost predictions for (a) quadrotor system and (b) Puma 560 system. Error bars represent standard deviations over 10 independent rollouts. (c)-(d): Comparison of computation time on a log scale between (c) SSGP-Lin and GP-EMM; (d) SSGP-EMM and GP-EMM. The horizontal axis is the input and output dimension (equal in this case). Vertical axis is the CPU time in seconds.

### 3.5.1 Computational Efficiency

In terms of the computational demand, we tested the CPU time for one-step prediction using SSGP-EMM and SSGP-Lin and full GP-EMM. We used sets of 800 random data points of 1,10,20,30,40,50,60,70,80,90 and 100 dimensions to learn SSGP and GP models. The results are shown in Figure 3.1c, Figure 3.1d. Both SSGP-EMM and SSGP-Lin show significantly less computational demand than GP-EMM with similar prediction performance (Figure 3.1c,Figure 3.1d). Our methods are more scalable than GP-EMM, which is the major computational bottleneck for probabilistic model-based RL approaches [37, 41].

### 3.5.2 Accuracy of Multi-Step Prediction

We compare the proposed approximate inference methods with three existing approaches: the full GP exact moment matching (GP-EMM) approach [33, 34, 35], Subset of Regressors GP (SoR-GP) [48] used in AGP-iLQR [57], and LWPR [60] used in iLQG-LD [56]. Note that SoR-GP and LWPR do not take into account input uncertainty when performing regressions.

We consider two multi-step prediction tasks using the dynamics models of a quadrotor (16 state dimensions, 4 control dimensions) and a Puma-560 manipulator (12 state dimensions, 6 control dimensions), and evaluate the performance in terms of prediction accuracy. We collected training sets of 1000 and 2000 data points for the quadrotor and puma task, respectively, and used 100 and 50 random features for our methods. We used 100 and 50 reference points for SoR-GP. Based on the learned models, we used a set of 10 initial states and control sequences to perform rollouts (200 steps for quadrotor and 100 steps for Puma) and compute the cost expectations at each step. Appendix 3.1(a)(b) shows the prediction errors. It can be seen that SSGP-EMM is very close to GP-EMM and SSGP-EMM performs slightly better than SSGP-Lin in all cases. Since SoR-GP and LWPR do not take into account input uncertainty when performing regression, our methods outperform them consistently.

### 3.5.3 Model Predictive Control

*Tracking a Moving Target*

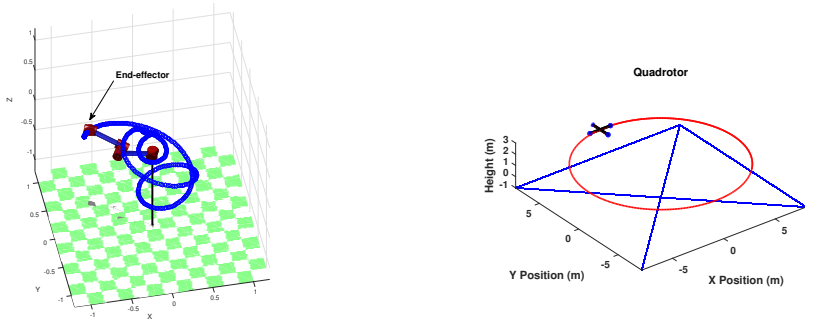
We consider the Puma-560 manipulator and quadrotor systems with dynamics model specified by SSGPs. For both tasks the goal is to track a moving target. In addition, the true system dynamics vary online, which necessitates both online optimization and model update, as we do here. The details of the tasks and training the SSGP dynamics models are provide below.

**PUMA-560 task: moving target and model parameter changes** The task is to steer the end-effector to the desired position and orientation. The desired state is time-varying over 800 time steps as shown in Figure 3.2a. We collected 1000 data points offline and sampled 50 random features for both of our methods. Similarly for AGP-iLQR we used 50 reference points. In order to show the effect of online adaptation, we increased the mass of the end-effector by 500% at the beginning of online learning (it is fixed during learning).

**Quadrotor task: time-varying tasks and dynamics** The objective is to start at (-1, 1, 0.5) and track a moving target as shown in Figure 3.2b for 400 steps. The mass of the quadrotor is decreasing at a rate of 0.02 kg/step. The controls are thrust forces of the 4 rotors and we consider the control constraint $u_{\min} = 0.5, u_{\max} = 3$. We collected 3000 data points offline, and sampled 100 and 400 features for online learning. The forgetting factor for online learning $\lambda = 0.992$.

Results in terms of cost $l(x_t, u_t)$ are shown in Figure 3.4. Figure 3.4a shows that our methods outperform iLQG-LD [56] and AGP-iLQR [57]. The similarities and differences between these methods have been discussed in $\S Section\ 3.4$. Figure 3.4a show that our method based on SSGP-EMM slightly outperforms the SSGP-lin based method, and both of them have better performance than iLQG-LD and AGP-iLQG that do not consider model uncertainty. In Figure 3.4b, SSGP-Lin was used for approximate inference and the receding-

horizon DDP (RH-DDP) [52] with full knowledge of the dynamics model was used as a baseline. The figure shows that model update is necessary and more features could improve performance.




(a) Puma 560 task (b) Quadrotor task

Figure 3.2: PUMA-560 and quadrotor tasks

*Autonomous Drifting: Steady-State Stabilization*

In this example, we study the control of a wheeled vehicle during extreme operation conditions (powerslide). The task is to stabilize the vehicle to a specified steady-state using purely longitudinal control. The desired steady-state consists of velocity $V$ , side slip angle $\beta$, and yaw rate $\frac{V}{R}$ where $R$ is the path radius. This problem has been studied in [61] where the authors developed a LQR control scheme based on analytic linearization of the dynamics model. However, this method is restrictive due to the assumption of full knowledge of the complex dynamics model.

We applied our method to this task without any prior model knowledge with 2500 offline data points, which were sampled from the empirical vehicle model in [61]. We used 50, 150, and 400 random features and SSGP-Lin for approximate inference in our experiments. Results and comparison to [61] are illustrated in Figure 3.3. Figure 3.3 shows that performance improves with a larger number of features, and with a moderate number of features, MPC with SSGP-Lin behaves very closely to the ground truth solution.

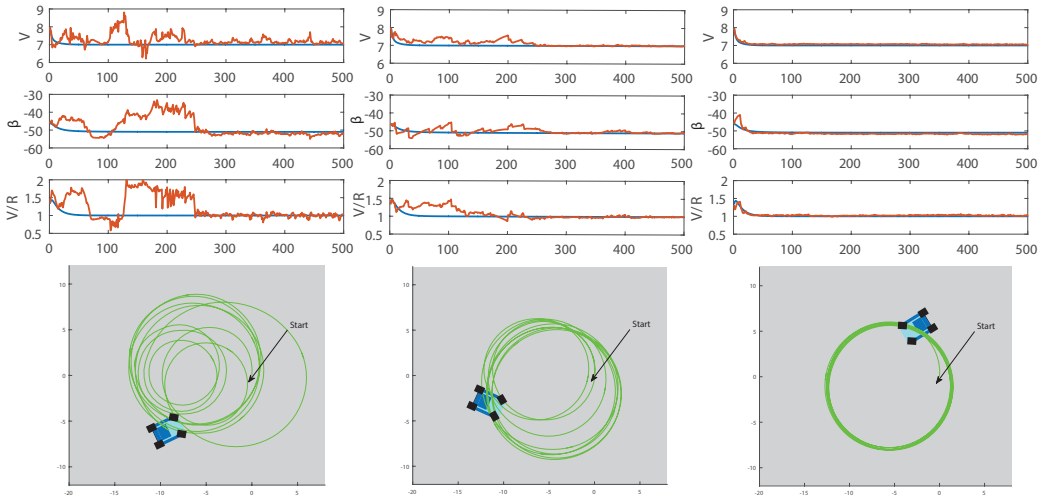
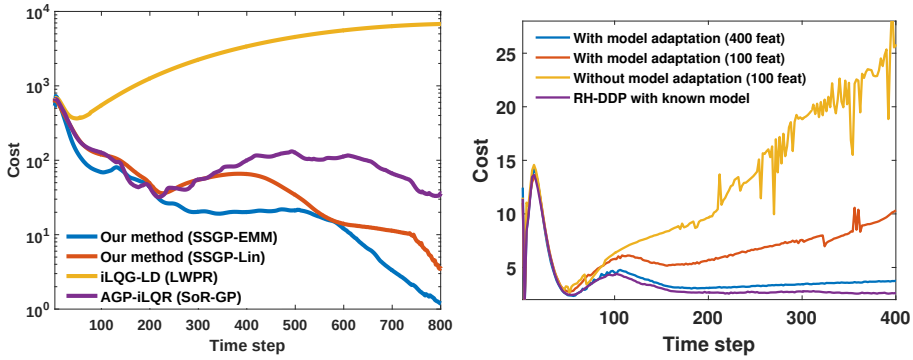



Figure 3.3: Comparison of the drifting performance using 50 (left), 150 (middle) and 400 (right) random features. Blue lines are the solution provided in [61].




(a) Robotic arm task cost (b) Quadrotor task cost

Figure 3.4: Cost comparison for arm and quadrotor tasks. In The results are the average over 3 independent trials.

## 3.6 Conclusion

Motivated by the high computation complexity of using full Gaussian processes in model predictive control, we introduced two analytic moment-based approaches to *predicting under uncertainty* in sparse spectrum Gaussian processes (SSGPs). Compared to their full GP counterparts, our methods are more general: they are applicable to any continuous shift-invariant kernel. They also scale to larger datasets by leveraging random features with frequencies sampled from the spectral density of a given kernel (see Table 3.1, Table 3.2).

Although we adopt the name SSGP, our proposed methods are not tied to specific model learning methods such as linear Bayesian regression [44]. They can be applied to any SSGP with a specified feature weight distribution (3.5), and $\alpha$ and $A$ can be computed via different approaches. For example, $A$ can be iteratively computed by methods like doubly stochastic gradient descent [46].

The proposed methods are verified in solving the model predictive control problem. Our methods directly address the challenging aspects of model predictive control: model uncertainty and real-time execution constraints. We evaluated our algorithm on extensive simulated examples and showed that SSGP-EMM (Section 3.3.1) and SSGP-Lin (Section 3.3.2) are accurate alternatives to their full GP counterparts when learning from large amounts of data.

Figure 3.5: The AutoRally car and the test track.

Moreover, our SSGP-based MPC algorithm has been verified on a real-world autonomous high-speed off-road driving task Figure 3.5. It achieved high real-time performance by solving the optimization (3.18) in MPC at frequency 50Hz. It was used as an algorithmic expert policy in one of our online imitation learning project [7]

# Part II

# Unbiased Policy Optimization

# CHAPTER 4
# PREDICTABLE ONLINE POLICY OPTIMIZATION

## 4.1 Introduction

In Chapter 2 and Chapter 3, we addressed the computational burden of using stochastic continuous-time linear dynamics models in solving the trajectory estimation and mapping problem and using Gaussian process dynamics model in solving the model predictive control problem. Our research largely improve the practicality of using these two powerful dynamics models in a challenging setting where the trajectory optimization operation is performed in *every time step*.

In this chapter and Chapter 6, we will focus on accelerating policy optimization using dynamics models. Instead of optimizing over an open-loop state or control sequence, policy optimization involves reactive policies that map from state or observation to (distribution of) action. Furthermore, we will consider an episodic on-policy setting. In this setting, learner's policy is updated in an iterative fashion: in each round/iteration, the learner gathers data by executing its current policy for several episodes in the environment and uses the data to update its policy. Because under this episodic setting, the policy is updated offline after data are collected, computation complexity will not be our concern. However, since the process of data collection is time-consuming and labor-intensive, developing theories and algorithms that can help improve the sample efficiency of learning is our goal.

To address the critical sample complexity issue in policy optimization, model-based policy optimization methods improve sample efficiency by leveraging an accurate dynamics model that can cheaply simulate interactions to compute policy updates in lieu of real-world interactions [62, 63, 64, 65, 41, 66, 67, 68, 69, 70, 71, 72]. However, all of these approaches, while potentially accelerating policy learning, suffer from a common drawback: when

the model is inaccurate, the performance of the policy can become *biased* away from the best achievable in the policy class. Several strategies have been proposed to remove this performance bias. Learning-to-plan attempts to train the planning process end-to-end [73, 74, 75], so the performance of a given planning structure is directly optimized. However, these algorithms are still optimized through standard model-free techniques; it is unclear as to whether they are more sample efficient.

In this chapter, we provide a novel learning framework that can leverage dynamics models or, more general, predictive models (Section 4.4.2) to improve sample efficiency while avoiding performance bias due to modeling errors for solving two important policy optimization problems: imitation learning (IL) and reinforcement learning (RL).

Our approach is built on techniques from online learning [76, 77]. The use of online learning to analyze policy optimization was pioneered by Ross et al. [78], who proposed to reduce IL to adversarial online learning. This reduction provides a framework for performance analysis, leading to algorithms such as DAGGER [78] and AGGREVATE [79]. However, it was recently shown that the naïve reduction to adversarial online learning loses information [80]: in practice, IL is *predictable* [9] and can be thought of as a predictable online learning problem [81]. Based on this insight, we develop a novel first-order learning framework, PICCOLO (PredICtor-COrrector poLicy Optimization), for general predictable online learning problems. PICCOLO is a *meta-algorithm*: it takes a standard online learning algorithm designed for adversarial problems (e.g.,ADAGRAD [82]) as input and returns a new hybrid algorithm that can use model information to accelerate convergence.

Our other contributions in this chapter include:

1. We show that RL can also be formulated as a predictable online learning problem which largely widens the scope of problems that PICCOLO is applicable to solve.

2. We develop critical extensions of PICCOLO to handle nonlinear loss functions directly, instead of their first-order approximation to accelerate the existing IL and RL algorithms that do not linearize loss functions, such as DAGGER [78] and PPO [83].

## 4.2 Problem Definition

We consider solving policy optimization problems: given state and action spaces $\mathbb{S}$ and $\mathbb{A}$, and a parametric policy class $\Pi$, we desire a stationary policy $\pi \in \Pi$ that solves

$$\min_{\pi \in \Pi} J(\pi), \quad J(\pi) := \mathbb{E}_{(s,t)\sim d_\pi} \mathbb{E}_{a \sim \pi_s} [c_t(s,a)] \tag{4.1}$$

where $c_t(s,a)$ is the instantaneous cost at time $t$ of state $s \in \mathbb{S}$ and $a \in \mathbb{A}$, $\pi_s$ is the distribution of $a$ at state $s$ under policy $\pi$, and $d_\pi$ is a generalized stationary distribution of states generated by running policy $\pi$ in a Markov decision process (MDP); the notation $\mathbb{E}_{a\sim\pi_s}$ denotes evaluation when $\pi$ is deterministic. The use of $d_\pi$ in (4.1) abstracts different discrete-time RL/IL problems into a common setup. For example, an infinite-horizon $\gamma$-discounted problem with time-invariant cost $c$ can be modeled by setting $c_t = c$ and $d_\pi(s,t) = (1-\gamma)\gamma^t d_{\pi,t}(s)$, where $d_{\pi,t}$ is the state distribution visited by policy $\pi$ at time $t$ starting from some *fixed* but unknown initial state distribution.

## 4.3 IL and RL as Predictable Online Learning

We study policy optimization through the lens of online learning [84], by treating a policy optimization algorithm as the learner in online learning and *each intermediate policy* that it produces as an online decision. This identification recasts the iterative process of policy optimization into a standard online learning setup: in round $n$, the learner plays a decision $\pi_n \in \Pi$, a *per-round loss* $l_n$ is then selected, and finally some information of $l_n$ is revealed to the leaner for making the next decision. We note that the "rounds" considered here are the number of episodes that an algorithm interacts with the (unknown) MDP environment to obtain new information, not the time steps in the MDP. And we will suppose the learner receives an unbiased stochastic approximation $\tilde{l}_n$ of $l_n$ as feedback.

We show that, when the per-round losses $\{l_n\}$ are properly selected, the policy performance $\{J(\pi_n)\}$ in IL and RL can be upper bounded in terms the $N$-round weighted

regret

$$\text{Regret}_N(l) := \sum_{n=1}^{N} w_n l_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{N} w_n l_n(\pi) \tag{4.2}$$

and an expressiveness measure of the policy class $\Pi$

$$\epsilon_{\Pi,N}(l) := \frac{1}{w_{1:N}} \min_{\pi \in \Pi} \sum_{n=1}^{N} w_n l_n(\pi) \tag{4.3}$$

where $w_n > 0$ and $w_{1:n} := \sum_{m=1}^{n} w_m$. Moreover, we show that these online learning problems are *predictable*: that is, the per-round losses are not completely adversarial but can be estimated from past information. We will use these ideas to design PICCOLO in the next section.

### 4.3.1 IL as Online Learning

We start by reviewing the classical online learning approach to IL (online IL for short) [78] to highlight some key ideas. IL leverages domain knowledge about a policy optimization problem through expert demonstrations. Online IL, in particular, optimizes policies by letting the learner $\pi$ query the expert $\pi^\star$ for desired actions, so that a policy can be quickly trained to perform as well as the expert. At its heart, online IL is based on the following lemma, which relates the performance between $\pi$ and $\pi^\star$.

**Lemma 1.** [85] *Let $\pi$ and $\pi'$ be two policies and $A_{\pi',t}(s,a) := Q_{\pi',t}(s,a) - V_{\pi',t}(s)$. Then $J(\pi) = J(\pi') + \mathbb{E}_{d_\pi}\mathbb{E}_\pi[A_{\pi'}]$.*

Given the equality in Lemma 1, the performance difference between $\pi$ and $\pi^\star$ can then be upper-bounded as

$$J(\pi) - J(\pi^\star) = \mathbb{E}_{d_\pi}\mathbb{E}_\pi[A_{\pi^\star}] \leq C_{\pi^\star}\mathbb{E}_{(s,t)\sim d_\pi}[D_t(\pi^\star_s || \pi_s)]$$

for some positive constant $C_{\pi^\star}$ and function $D_t$, which is often derived from statistical

distances such as KL divergence [86]. When $A_{\pi^\star,t}$ is available, we can also set $D_t(\pi_s^\star||\pi_s) = \mathbb{E}_{a\sim\pi_s}[A_{\pi^\star,t}(s,a)]$, as in value aggregation (AGGREVATE) [79].

Without loss of generality, let us suppose $D_t(\pi_s^\star||\pi_s) = \mathbb{E}_{a\sim\pi_s}[\bar{c}_t(s,a)]$ for some $\bar{c}_t$. Online IL converts policy optimization into online learning with per-round loss

$$l_n(\pi) := \mathbb{E}_{d_{\pi_n}}\mathbb{E}_\pi[\bar{c}]. \tag{4.4}$$

By the inequality above, it holds that $J(\pi_n) - J(\pi^\star) \leq C_{\pi^\star} l_n(\pi_n)$ for every $n$, establishing the reduction below.

**Lemma 2.** [9] *For $l_n$ defined in* (4.4)*, we have*

$$\mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n J(\pi_n)}{w_{1:N}}\right] \leq J(\pi^\star) + C_{\pi^\star}\mathbb{E}\left[\epsilon_{\Pi,N}(l) + \frac{\mathrm{Regret}_N(\tilde{l})}{w_{1:N}}\right]$$

*where the expectation is due to sampling $\tilde{l}_n$.*

That is, when a no-regret algorithm is used, the performance concentrates toward $J(\pi^\star) + C_{\pi^\star}\mathbb{E}[\epsilon_{\Pi,N}(l)]$.

### 4.3.2 RL as Online Learning

Can we also formulate RL as online learning? Here we propose a new perspective on RL using Lemma 1. Given a policy $\pi_n$ in round $n$, we define a per-round loss

$$l_n(\pi) := \mathbb{E}_{d_{\pi_n}}\mathbb{E}_\pi[A_{\pi_{n-1}}]. \tag{4.5}$$

which describes how well a policy $\pi$ performs relative to the previous policy $\pi_{n-1}$ under the state distribution of $\pi_n$. By Lemma 1, for $l_n$ defined in (4.5), $l_n(\pi_n) = J(\pi_n) - J(\pi_{n-1})$ for every $n$, similar to the pointwise inequality of $l_n$ that Lemma 2 is based on. With this observation, we derive the reduction below (proved in Appendix B.2).

**Lemma 3.** *Suppose* $\frac{w_{n+k}}{w_n} \leq \frac{w_{m+k}}{w_m}$*, for all* $n \geq m \geq 1$ *and* $k \geq 0$*. For* (4.5) *and any* $\pi_0$*,* $\mathbb{E}[\sum_{n=1}^{N} \frac{w_n J(\pi_n)}{w_{1:N}}] \leq J(\pi_0) + \sum_{n=1}^{N} \frac{w_{N-n+1}}{w_{1:N}} \mathbb{E}[\mathrm{Regret}_n(\tilde{l}) + w_{1:n}\epsilon_{\Pi,n}(l)]$*, where the expectation is due to sampling* $\tilde{l}_n$*.*

*Interpretations*

Lemma 3 is a policy improvement lemma, which shows that when the learning algorithm is no-regret, the policy sequence improves on-average from the initial reference policy $\pi_0$ that defines $l_1$. This is attributed to an important property of the definition in (4.5) that $\min_{\pi\in\Pi} l_n(\pi) \leq 0$. To see this, suppose $\mathbb{E}[\epsilon_{\Pi,n}(l)] \leq -\Omega(1)$ (i.e.,there is a policy that is better than all previous $n$ policies); this is true for small $n$ or when the policy sequence is concentrated. Under this assumption, if $w_n = 1$ and $\mathrm{Regret}_n(\tilde{l}) \leq O(\sqrt{n})$, then the average performance improves roughly $N\mathbb{E}[\epsilon_{\Pi,N}(l)]$ away from $J(\pi_0)$.

While it is unrealistic to expect $\mathbb{E}[\epsilon_{\Pi,n}(l)] \leq 0$ for large $n$, we can still use Lemma 3 to comprehend *global* properties of policy improvement, for two reasons. First, the inequality in Lemma 3 holds for any interval of the policy sequence. Second, as we show in Appendix B.2, the Lemma 3 also applies to dynamic regret [77], with respect to which $\mathbb{E}[\epsilon_{\Pi,n}(l)]$ is always negative. Therefore, if an algorithm is strongly-adaptive [87] (i.e.,it is no-regret for any interval) or has sublinear dynamic regret [88], then its generated policy sequence will strictly, non-asymptotically improve. In other words, for algorithms with a stronger notion of convergence, Lemma 3 describes the global improvement rate.

The choice of per-round loss in (4.5) has an interesting relationship to actor-critic in RL [89]. Although actor-critic methods, theoretically, use $\mathbb{E}_{d_{\pi_n}}(\nabla\mathbb{E}_\pi)[A_{\pi_n}]|_{\pi=\pi_n}$ to update policy $\pi_n$, in practice, they use $\mathbb{E}_{d_{\pi_n}}(\nabla\mathbb{E}_\pi)[A_{\pi_{n-1}}]|_{\pi=\pi_n}$, because the advantage/value function estimate in round $n$ is updated after the policy update in order to prevent bias due to over-fitting on finite samples [90]. This practical gradient is *exactly* $\nabla\tilde{l}_n(\pi_n)$, the sampled gradient of (4.5). Therefore, Lemma 3 explains the properties of these practical modifications.

#### 4.3.3 Predictability

An important property of the above online learning problems is that they are not completely adversarial [80]. This can be seen from the definitions of $l_n$ in (4.4) and (4.5), respectively. For example, suppose the cost $c_t$ in the original RL problem (4.1) is known; then the information unknown before playing the decision $\pi_n$ in the environment is only the state distribution $d_{\pi_n}$. Therefore, the per-round loss cannot be truly adversarial, as the same dynamics and cost functions are used across different rounds. That is, in an idealized case where the true dynamics and cost functions are exactly known, using the policy returned from a model-based RL algorithm would incur zero regret, since only the interactions with the real MDP environment, not the model, counts as rounds. We will exploit this property to design PICCOLO.

### 4.4 Predictor-Corrector Learning

We showed that the performance of RL and IL can be bounded by the regret of properly constructed predictable online learning problems. These results provide a foundation for designing policy optimization algorithms: efficient learning algorithms for policy optimization can be constructed from powerful online learning algorithms that achieve small regret. This perspective explains why common methods (e.g.,mirror descent) based on gradients of (4.4) and (4.5) work well in IL and RL. However, the predictable nature of policy optimization problems suggests that directly applying these standard online learning algorithms designed for adversarial settings is *suboptimal*. The predictable information must be considered to achieve optimal convergence.

One way to include predictable information is to develop specialized two-step algorithms based on, e.g., mirror-prox or FTRL-prediction [91, 81, 92]. For IL, MOBIL was recently proposed [9], which updates policies by approximate Be-the-Leader [93] and provably achieves faster convergence than previous methods. However, these two-step algorithms

often have obscure and non-sequential update rules, and their adaptive and accelerated versions are less accessible [94]. This can make it difficult to implement and tune them in practice.

Here we take an alternative, *reduction-based* approach. We present PICCOLO, a general first-order framework for solving predictable online learning problems. PICCOLO is a meta-algorithm that turns a base algorithm designed for adversarial problems into a new algorithm that can leverage the predictable information to achieve better performance. As a result, we can adopt sophisticated first-order adaptive algorithms to optimally learn policies, without reinventing the wheel. Specifically, given *any* first-order base algorithm belonging to the family of (adaptive) mirror descent and FTRL algorithms, we show how one can "PICCOLO it" to achieve a faster convergence rate without introducing additional performance bias due to prediction errors. Most first-order policy optimization algorithms belong to this family [86], so we can PICCOLO these model-free algorithms into new hybrid algorithms that can robustly use (imperfect) predictive models, such as off-policy gradients and simulated gradients, to improve policy learning.

### 4.4.1 The PICCOLO Idea

The design of PICCOLO is based on the observation that an $N$-round predictable online learning problem can be written as a new adversarial problems with $2N$ rounds. To see this, let $\{l_n\}_{n=1}^N$ be the original predictable loss sequence. Suppose, before observing $l_n$, we have access to a *model loss* $\hat{l}_n$ that contains the predictable information of $l_n$. Define $\delta_n = l_n - \hat{l}_n$. We can then write the accumulated loss (which regret concerns) as $\sum_{n=1}^N l_n(\pi_n) = \sum_{n=1}^N \hat{l}_n(\pi_n) + \delta_n(\pi_n)$. That is, we can view the predictable problem with $\{l_n\}_{n=1}^N$ as a new adversarial online learning problem with a loss sequence $\hat{l}_1, \delta_1, \hat{l}_2, \delta_2, \ldots, \hat{l}_N, \delta_N$.

The idea of PICCOLO is to apply standard online learning algorithms designed for adversarial settings to this new $2N$-round problem. This would create a new set of decision variables $\{\hat{\pi}_n\}_{n=1}^N$, in which $\hat{\pi}_n$ denotes the decision made before seeing $\hat{l}_n$, and leads to

the following sequence $\pi_1, \delta_1, \hat{\pi}_2, \hat{l}_2, \pi_2, \delta_2, \dots$ (in which we define $\delta_1 = l_1$). We show that when the base algorithm is optimal in adversarial settings, this simple strategy results in a decision sequence $\{\pi_n\}_{n=1}^N$ whose regret with respect to $\{l_n\}_{n=1}^N$ is optimal, just as those specialized two-step algorithms [91, 81, 92]. In Appendix B.1, we show PICCOLO unifies and generalize these two-step algorithms to be adaptive.

### 4.4.2 The Meta Algorithm PICCOLO

We provide details to realize this reduction. We suppose, in round $n$, the model loss is given as $\hat{l}_n(\pi) = \langle \hat{g}_n, \pi \rangle$ for some vector $\hat{g}_n$, and stochastic first-order feedback $g_n = \nabla \tilde{l}_n(\pi_n)$ from $l_n$ is received.

*Base Algorithms*

We first give a single description of different base algorithms for the formal definition of the reduction steps. Here we limit our discussions to mirror descent and postpone the FTRL case to Appendix B.3. We assume that $\Pi$ is a convex compact subset in some normed space with norm $\|\cdot\|$, and we use $B_R(\pi||\pi') = R(\pi) - R(\pi') - \langle \nabla R(\pi'), \pi - \pi' \rangle$ to denote a Bregman divergence generated by a strictly convex function $R$, called the distance generator.

Mirror descent updates decisions based on proximal maps. In round $n$, given direction $g_n$ and weight $w_n$, it executes

$$\pi_{n+1} = \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + B_{R_n}(\pi||\pi_n) \tag{4.6}$$

where $R_n$ is a strongly convex function; (4.6) reduces to gradient descent with step size $\eta_n$ when $R_n(\cdot) = \frac{1}{2\eta_n}\|\cdot\|^2$. More precisely, (4.6) is composed of two steps: *1)* the update of the distance generator to $R_n$, and *2)* the update of the decision to $\pi_{n+1}$; different mirror descent algorithms differ in how the regularization is selected and adapted.

PICCOLO explicitly treats a base algorithm as the composition of two basic operations

(this applies also to FTRL)

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, H_{n-1}, g_n, w_n) \\ h_{n+1} &= \texttt{update}(h_n, H_n, g_n, w_n) \end{aligned} \tag{4.7}$$

so that later it can recompose them to generate the new algorithm. For generality, we use $h$ and $H$ to denote the abstract representations of the decision variable and the regularization, respectively. In mirror descent, $h$ is exactly the decision variable, $H$ is the distance generator, and we can write $\texttt{update}(h, H, g, w) = \arg\min_{\pi' \in \Pi} \langle wg, \pi' \rangle + B_H(\pi' || h)$. The operation `adapt` denotes the algorithm-specific scheme for the regularization update (e.g.,changing the step size), which in general updates the size of regularization to grow slowly and inversely proportional to the norm of $g_n$.

*The* PICCOLO*ed Algorithm*

PICCOLO generates decisions by applying a given base algorithm in (4.7) to the new problem with losses $\delta_1, \hat{l}_2, \delta_2, \ldots$. This is accomplished by recomposing the basic operations in (4.7) into the Prediction and the Correction Steps:

$$h_n = \texttt{update}(\hat{h}_n, H_{n-1}, \hat{g}_n, w_n) \qquad \text{[Prediction]}$$

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, H_{n-1}, e_n, w_n) \\ \hat{h}_{n+1} &= \texttt{update}(h_n, H_n, e_n, w_n) \end{aligned} \qquad \text{[Correction]}$$

where $\hat{h}_n$ is the abstract representation of $\hat{\pi}_n$, and $e_n = g_n - \hat{g}_n$ is the error direction. We can see that the Prediction and Correction Steps are exactly the update rules resulting from applying (4.7) to the new adversarial problem, except that only $h_n$ is updated in the Prediction Step, *not* the regularization (i.e.,the step size). This asymmetry design is important for achieving optimal regret, because in the end we care only about the regret of $\{\pi_n\}$ on the original loss sequence $\{l_n\}$.

In round $n$, the "PICCOLOed" algorithm first performs the Prediction Step using $\hat{g}_n$ to generate the learner's decision (i.e.,$\pi_n$) and runs this new policy in the environment to get the true gradient $g_n$. Using this feedback, the algorithm performs the Correction Step to amend the bias of using $\hat{g}_n$. This is done by first adapting the regularization to $H_n$ and then updating $\pi_n$ to $\hat{\pi}_{n+1}$ along the error $e_n = g_n - \hat{g}_n$.

*Model Losses and Predictive Models*

The Prediction Step of PICCOLO relies on the vector $\hat{g}_n$ to approximate the future gradient $g_n$. Here we discuss different ways to specify $\hat{g}_n$ based on the concept of predictive models [9]. A *predictive model* $\Phi_n$ is a first-order oracle such that $\Phi_n(\cdot)$ approximates $\nabla l_n(\cdot)$. In practice, a predictive model can be a simulator with an (online learned) dynamics model [62, 65], or a neural network trained to predict the required gradients [66, 67]. An even simpler heuristic is to construct predictive models by *off-policy* gradients $\Phi_n(\cdot) = \sum_{m=n-K}^{n-1} \nabla \tilde{l}_m(\cdot)$ where $K$ is the buffer size.

In general, we wish to set $\hat{g}_n$ to be close to $g_n$, as we will later show in Section 4.5 that the convergence rate of PICCOLO depends on their distance. However, even when we have perfect predictive models, this is still a non-trivial task. We face a chicken-or-the-egg problem: $g_n$ depends on $\pi_n$, which in turn depends on $\hat{g}_n$ from the Prediction Step. Therefore, we propose to solve for $\hat{g}_n$ and $\pi_n$ *simultaneously*. That is, we wish to solve a fixed-point problem, finding $h_n$ such that

$$h_n = \texttt{update}(\hat{h}_n, H_{n-1}, \Phi_n(\pi_n(h_n)), w_n) \tag{4.8}$$

The exact formulation of the fixed-point problem depends on the class of base algorithms. For mirror descent, it is a variational inequality: find $\pi_n \in \Pi$ such that $\forall \pi \in \Pi$, $\langle \Phi_n(\pi_n) + \nabla R_{n-1}(\pi_n) - \nabla R_{n-1}(\hat{\pi}_n), \pi - \pi_n \rangle \geq 0$. In a special case when $\Phi_n = \nabla f_n$ for some function $f_n$, the above variational inequality is equivalent to finding a stationary point

of the optimization problem $\min_{\pi\in\Pi} f_n(\pi) + B_{R_{n-1}}(\pi||\hat{\pi}_n)$. In other words, one way to implement the Prediction Step is to solve the above minimization problem for $\pi_n$ and use $\nabla f_n(\pi_n)$ as the effective prediction $\hat{g}_n$.

**Algorithm 3** PICCOLO

**Input:** policy $\pi_1$, cost sequence $\{\psi_n\}$, regularization $H_0$, model $\Phi_1$, iteration $N$, exponent $p$
**Output:** $\bar{\pi}_N$

1: Set $\hat{\pi}_1 = \pi_1$ and weights $w_n = n^p$
2: Sample integer $K \in [1, N]$ with $P(K = n) \propto w_n$
3: **for** $n = 1 \dots K - 1^*$ **do**
4: $\quad \pi_n, \hat{g}_n = \texttt{PredictionStep}(\hat{\pi}_n, \Phi_n, H_{n-1}, w_n)$
5: $\quad \mathcal{D}_n, g_n = \texttt{DataCollection}(\pi_n)$
6: $\quad H_n, \hat{\pi}_{n+1} = \texttt{CorrectionStep}(\pi_n, e_n, H_{n-1}, w_n)$, where $e_n = g_n - \hat{g}_n$.
7: $\quad \Phi_{n+1} = \texttt{ModelUpdate}(\Phi_n, \mathcal{D})$, where $\mathcal{D} = \mathcal{D} \bigcup \mathcal{D}_n$.
8: **end for**
9: Set $\bar{\pi}_N = \pi_{K-1}$

### 4.4.3 Summary: Why Does PICCOLO Work?

We provide a summary of the full algorithm for policy optimization in Algorithm 3. We see that PICCOLO uses the predicted gradient to take an extra step to accelerate learning, and, meanwhile, to prevent the error accumulation, it adaptively adjusts the step size (i.e.,the regularization) based on the prediction error and corrects for the bias on the policy right away. To gain some intuition, let us consider ADAGRAD [82] as a base algorithm[†]:

$$G_n = G_{n-1} + \text{diag}(w_n g_n \odot w_n g_n)$$
$$\pi_{n+1} = \arg\min_{\pi\in\Pi} \langle w_n g_n, \pi\rangle + \frac{1}{2\eta}(\pi - \pi_n)^\top G_n^{1/2}(\pi - \pi_n)$$

where $G_0 = \epsilon I$ and $\eta, \epsilon > 0$, and $\odot$ denotes element-wise multiplication. This update has an `adapt` operation as $\texttt{adapt}(h, H, g, w) = G + \text{diag}(wg \odot wg)$ which updates the Bregman divergence based on the gradient size.

PICCOLO transforms ADAGRAD into a new algorithm. In the Prediction Step, it

[†]We provide another example in Appendix B.5.

performs

$$\pi_n = \arg\min_{\pi \in \Pi} \langle w_n \hat{g}_n, \pi \rangle + \frac{1}{2\eta}(\pi - \pi_{n-1})^\top G_{n-1}^{1/2}(\pi - \pi_{n-1})$$

In the Correction Step, it performs

$$G_n = G_{n-1} + \mathrm{diag}(w_n e_n \odot w_n e_n)$$
$$\hat{\pi}_{n+1} = \arg\min_{\pi \in \Pi} \langle w_n e_n, \pi \rangle + \frac{1}{2\eta}(\pi - \hat{\pi}_n)^\top G_n^{1/2}(\pi - \hat{\pi}_n)$$

We see that the PICCOLO-ADAGRAD updates $G_n$ proportional to the prediction error $e_n$ instead of $g_n$. It takes larger steps when models are accurate, and decreases the step size once the prediction deviates. As a result, PICCOLO is robust to model quality: it accelerates learning when the model is informative, and prevents inaccurate (potentially adversarial) models from hurting the policy. We will further demonstrate this in theory and in the experiments.

### 4.5 Theoretical Analysis of PICCOLO

In this section, we show that PICCOLO has two major benefits over previous approaches: *1)* it accelerates policy learning when the models predict the required gradient well on average; and *2)* it does not bias the performance of the policy, even when the prediction is incorrect.

To analyze PICCOLO, we introduce an assumption to quantify the `adapt` operator of a base algorithm.

**Assumption 1.** `adapt` chooses a regularization sequence such that, for some $M_N = o(w_{1:N})$, $\|H_0\|_{\mathcal{R}} + \sum_{n=1}^{N} \|H_n - H_{n-1}\|_{\mathcal{R}} \leq M_N$ for some norm $\|\cdot\|_{\mathcal{R}}$ which measures the size of regularization.

This assumption, which requires the regularization to increase slower than the growth of $w_{1:N}$, is satisfied by most reasonably-designed base algorithms. For example, in a uniformly

weighted problem, gradient descent with a decaying step size $O(\frac{1}{\sqrt{n}})$ has $M_N = O(\sqrt{N})$. In general, for stochastic problems, an optimal base algorithm would ensure $M_N = O(\frac{w_{1:N}}{\sqrt{N}})$.

### 4.5.1 Convergence Properties

Now we state the main result, which quantifies the regret of PICCOLO with respect to the sequence of linear loss functions that it has access to. The proof is given in Appendix B.6.

**Theorem 1.** *Suppose $H_n$ defines a strongly convex function with respect to $\|\cdot\|_n$. Under Assumption 1, running* PICCOLO *ensures $\sum_{n=1}^{N} \langle w_n g_n, \pi_n - \pi \rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$, for all $\pi \in \Pi$.*

The term $\|e_n\|_{*,n}^2$ in Theorem 1 says that the performance of PICCOLO depends on how well the base algorithm adapts to the error $e_n$ through the `adapt` operation in the Correction Step. Usually `adapt` updates $H_n$ gradually (Assumption 1) while minimizing $\frac{1}{2}\|e_n\|_{*,n}^2$, like we showed in ADAGRAD.

In general, when the base algorithm is adaptive and optimal for adversarial problems, we show in Appendix B.7 that its PICCOLOed version guarantees that

$$\mathbb{E}[\sum_{n=1}^{N} \langle w_n g_n, \pi_n - \pi \rangle] \leq O(1) + C_{\Pi,\Phi} \frac{w_{1:N}}{\sqrt{N}}$$

where $C_{\Pi,\Phi} = O(|\Pi| + E_\Phi + \sigma_g^2 + \sigma_{\hat{g}}^2)$ is some constant related to the diameter of $\Pi$ (denoted as $|\Pi|$), the model bias $E_\Phi$, and the sampling variance $\sigma_g^2$ and $\sigma_{\hat{g}}^2$ of $g_n$ and $\hat{g}_n$, respectively. Through Lemma 2 and Lemma 3, this bound directly implies accelerated and bias-free policy performance.

**Theorem 2.** *Suppose $\tilde{l}_n$ is convex*[‡] *and $w_n \geq \Omega(1)$. Then running* PICCOLO *yields $\mathbb{E}[\mathrm{Regret}_n(\tilde{l})/w_{1:N}] = O(\frac{C_{\Pi,\Phi}}{\sqrt{N}})$, where $C_{\Pi,\Phi} = O(|\Pi| + E_\Phi + \sigma_g^2 + \sigma_{\hat{g}}^2) = O(1)$.*

[‡]The convexity assumption is standard, as used in [82, 78, 95, 80], which holds for tabular problems as well as some special cases, like continuous-time problems (cf. [80]).

## 4.6 Extending PICCOLO to Nonlinear Loss Functions

We've focused on *linear* loss function and prediction function in both algorithm design and theoretical analysis of PICCOLO, due to the reason that dealing with linear loss function and prediction function is sufficient to representing predictable information and using it to accelerate learning as shown in Section 4.5.

However, in practice, IL and RL with nonlinear loss functions seem to achieve better performance. For example, DAGGER converges in 3 iterations on an high-speed autonomous car driving task [7] and RL algorithms that minimize a nonlinear surrogate function in each iteration (by running SGD for several epochs through data) obtain superior performance on simulated robot control and Atari environments [83]. Furthermore, because we consider policy optimization under the episodic setting in which the policy is updated offline given several episodes of data, optimization with nonlinear loss in each round does not impose a computational drawback. On the opposite, nonlinear losses capture more information than the linear ones on policy improvement. And this extra amount of information can lead to more sample efficient policy optimization.

In this section, we develop critical extensions of PICCOLO to handle nonlinear loss and prediction functions. Our extensions consist of two meta algorithms that transform online algorithms from the mirror descent and the FTRL family, respectively, to take advantage of predictable information.

### 4.6.1 Mirror Descent

First, we consider mirror descent as the base algorithm. PICCOLO deals with linear loss, i.e., $l_n(\pi) = \langle g_n, \pi \rangle$ for some $g_n$, and $\hat{l}_n(\pi) = \langle \hat{g}_n, \pi \rangle$ for some $\hat{g}_n$. In this case, the update

rule of PICCOLO can be written as

$$\pi_n = \arg\min_{\pi\in\Pi} \left\langle \nabla \hat{l}_n(\hat{\pi}_n), \pi \right\rangle + B_{H_{n-1}}(\pi||\hat{\pi}_n) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \arg\min_{\pi\in\Pi} \langle \nabla \delta_n(\pi_n), \pi \rangle + B_{H_n}(\pi||\pi_n) \qquad \text{[Correction]} \qquad (4.9)$$

where $H_n$ can be updated based on $e_n = \nabla\delta_n(\pi_n) = \nabla l_n(\pi_n) - \nabla\hat{l}_n(\hat{\pi}_n)$. Notice that in the Prediction Step, PICCOLO uses the regularization from the previous Correction Step.

In (4.9), we adopted a slightly different notation from Section 4.4, for a clear exposition of nonlinear PICCOLO which we will present next. PICCOLO with nonlinear loss does not assume that $l_n$ and $\hat{l}_n$ are linear. We show that its update rule can still be written using Prediction Step and Correction Step just as PICCOLO.

$$\pi_n = \arg\min_{\pi\in\Pi} \hat{l}_n(\pi) + B_{H_{n-1}}(\pi||\hat{\pi}_n) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \arg\min_{\pi\in\Pi} l_n(\pi) - \left\langle \nabla \hat{l}_n(\pi_n), \pi \right\rangle + B_{H_n}(\pi||\pi_n) \qquad \text{[Correction]}$$

where $H_n$ is updated based on $e_n = \nabla l_n(\pi_n) - \nabla\hat{l}_n(\pi_n)$. We show that the theoretical guarantees of linear PICCOLO described Theorem 1 and Theorem 2 hold similarly for this nonlinear PICCOLO algorithm. The proof can be adapted from the proof for these theorems with minor modifications by focusing on the nonlinear convex prediction error function $\delta_n(\pi) = l_n(\pi) - \langle \nabla\hat{l}_n(\pi_n), \pi\rangle$.

### 4.6.2 Follow-the-Regularized-Leader

Next consider another type of base algorithm, FTRL. When the losses are linear, FTRL is mainly different from mirror descent in the way that constrained decision sets are handled [96] . Whereas, when the loss functions are nonlinear, their memory and computation requirements are drastically different: the memory and computation of mirror descent remains constant, but they scale at least linearly with respect to number of iterations in

FTRL due to the need of storing and optimizing over the cumulative loss function. However, this does not annul the use of FTRL with nonlinear losses in policy optimization: in some policy optimization problems, sample complexity is a more dire concern than computation and space complexity. For instance, in online IL [78, 7], interacting with the environment and the expert is very expensive, which makes FTRL a preferred approach even though all previous data have to be stored and optimized over in each iteration.

This motivates us to accelerate nonlinear FTRL algorithms using predictable information based on the idea of PICCOLO. We start off by writing PICCOLO update rule with FTRL base algorithms:

$$\pi_n = \arg\min_{\pi\in\Pi} \langle w_n \hat{g}_n, \pi\rangle + \sum_{m=1}^{n-1} \langle w_m g_m, \pi\rangle + B_{r_m}(\pi||\pi_m) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \arg\min_{\pi\in\Pi} \sum_{m=1}^{n} \langle w_m g_m, \pi\rangle + B_{r_m}(\pi||\pi_m) \qquad \text{[Correction]}$$

where the regularization $r_m$ is updated using $e_n = g_n - \hat{g}_n$.

In nonlinear PICCOLO with FTRL base algorithms, instead of having two separate steps: Prediction Step and Correction Step, we update policy in one update step. This is due to the fact that all previous loss functions are saved without linearization. The update rule can be expressed as:

$$\pi_n = \arg\min_{\pi\in\Pi} \hat{l}_n(\pi) + \sum_{m=1}^{n-1} l_m(\pi) + B_{r_m}(\pi||\pi_m) \tag{4.10}$$

where the regularization $r_m$ is updated using $\nabla l_n(\pi_n) - \nabla\hat{l}_n(\pi)$. The convergence analysis results of linear PICCOLO in Theorem 1 and Theorem 2 hold similarly for this nonlinear PICCOLO algorithm. The proof is almost the same as the one for the linear case, except focusing on the nonlinear convex prediction error function $\delta_n(\pi) = l_n(\pi) - \langle\nabla\hat{l}_n(\pi_n), \pi\rangle$ and using the fact that $\pi_n$ computed using (4.10) also minimizes $\left\langle \nabla\hat{l}_n(\pi_n), \pi \right\rangle + \sum_{m=1}^{n-1} l_m(\pi) + B_{r_m}(\pi||\pi_m)$.

## 4.7 Experiments

We corroborate our theoretical findings with experiments[§] in learning neural network policies to solve robot RL tasks (CartPole, Hopper, Snake, and Walker3D) and IL tasks (CartPole, Reacher3D) from OpenAI Gym [97] with the DART physics engine [98].[¶] The aim is to see if PICCOLO improves the performance of a base algorithm, even though in these experiments the convexity assumption in the theory does not hold.

**Tasks** CartPole is a classic control problem, and its goal is to keep a pole balanced in a upright posture, by only applying force to the cart. Hopper, Snake, and Walker3D are locomotion tasks, of which the goal is to control an agent to move forward as quickly as possible without falling down (for Hopper and Walker3D) or deviating too much from moving forward (for Snake). Hopper is monopedal and Walker3D is bipedal, and both of them are subjected to significant contact discontinuities that are hard or even impossible to predict. Reacher3D is a manipulator reaching task, and its goal it to control a 5-DOF (degrees-of-freedom) manipulator to reach a random target position in a 3D space that is reset at the beginning of each episode.

### 4.7.1 Linear PICCOLO on RL Tasks

We first consider solving RL tasks through the online learning approach as described in Section 4.3.2. In particular, we choose several popular first-order mirror descent base algorithms in RL: ADAM [95], natural gradient descent NATGRAD [99], and trust-region optimizer TRPO [100]. We compute $g_n$ by GAE [101]. For predictive models, we consider off-policy gradients (with the samples of the last iteration LAST or a replay buffer REPLAY) and gradients computed through simulations with the true or biased dynamics models (TRUEDYN or BIASEDDYN). We will label a model with FP if $\hat{g}_n$ is determined by the

[§]The codes are available at `https://github.com/gtrll/rlfamily`.

[¶]The environments are defined in DartEnv, hosted at `https://github.com/DartEnv`.

fixed-point formulation (4.8)[‖]; otherwise, $\hat{g}_n = \Phi_n(\hat{\pi}_n)$. Please refer to Appendix B.8 for the details.

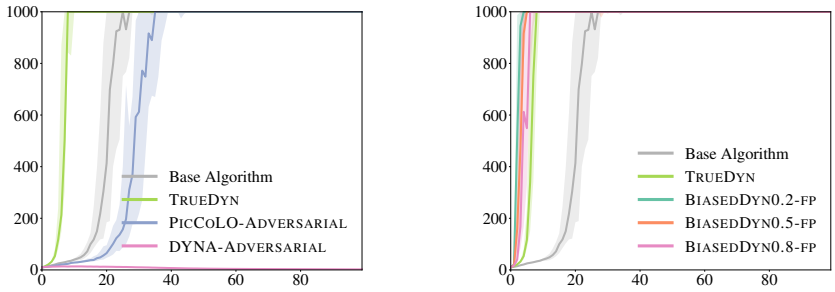
In Figure 4.1, we first use CartPole to study Theorem 2, which suggests that PICCOLO is unbiased and improves the performance when the prediction is accurate. Here we additionally consider an extremely bad model, ADVERSARIAL, that predicts the gradients adversarially.[**] Figure 4.1 (a) illustrates the performance of PICCOLO and DYNA, when ADAM is chosen as the base algorithm. We observe that PICCOLO improves the performance when the model is accurate (i.e.,TRUEDYN). Moreover, PICCOLO is robust to modeling errors. It still converges when the model is adversarially attacking the algorithm, whereas DYNA fails completely. In Figure 4.1 (b), we conduct a finer comparison of the effects of different model accuracies (BIASEDDYN-FP), when $\hat{g}_n$ is computed using (4.8). To realize inaccurate dynamics models to be used in the Prediction step, we change the mass of links of the robot by a certain factor, e.g.,BIASEDDYN0.8 indicates that the mass of each individual link is either increased or decreased by $80\%$ with probability 0.5, respectively. We see that the fixed-point formulation (4.8), which makes multiple queries of $\Phi_n$ for computing $\hat{g}_n$, performs much better than the heuristic of setting $\hat{g}_n = \Phi(\hat{\pi}_n)$, even when the latter is using the true model (TRUEDYN). Overall, we see PICCOLO with BIASEDDYN-FP is able to accelerate learning, though with a degree varying with model accuracies; but even for models with a large bias, it still converges unbiasedly, as we previously observed in Figure 4.1 (a), In Figure 4.2, we study the performance of PICCOLO in a range of environments. In general, we find that PICCOLO indeed improves the performance though the exact degree depends on how $\hat{g}_n$ is computed.[††] In Figure 4.2 (a) and (b), we show the results of using ADAM as the base algorithm. We observe that, while setting $\hat{g}_n = \Phi_n(\hat{\pi}_n)$ is already an effective heuristic, the performance of PICCOLO can be further and largely

[‖]In implementation, we solve the corresponding optimization problem with a few number of iterations. For example, BIASEDDYN-FP is approximately solved with 5 iterations.

[**]We set $\hat{g}_{n+1} = -\left(\max_{m=1,\ldots,n} \|g_m\|/\|g_n\|\right) g_n$.

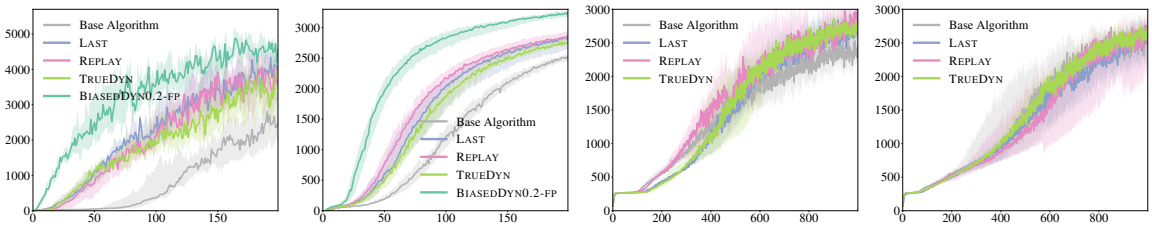
[††]Note that different base algorithms are not directly comparable, as further fine-tuning of step sizes is required.

improved if we adopt the fixed-point strategy in (4.8), as the latter allows the learner to take more globally informed update directions. Finally, to demonstrate the flexibility of the proposed framework, we also "PICCOLO" two other base algorithms, NATGRAD and TRPO, in Figure 4.2 (c) and (d), respectively. The complete set of experimental results can be found in Appendix B.8.




(a) Adversarial model (b) Different model fidelities

Figure 4.1: Performance of PICCOLO with different predictive models. $x$ axis is iteration number and $y$ axis is sum of rewards. The curves are the median among 8 runs with different seeds, and the shaded regions account for $25\%$ percentile. ADAM is used as the base algorithm, and the update rule, by default, is PICCOLO; e.g.,TRUEDYN in (a) refers to PICCOLO with TRUEDYN predictive model. (a) Comparison of PICCOLO and DYNA with adversarial model. (b) PICCOLO with the fixed-point setting (4.8) with dynamics model in different fidelities. BIASEDDYN0.8 indicates that the mass of each individual robot link is either increased or decreased by $80\%$ with probability 0.5 respectively.




(a) Hopper ADAM (b) Snake ADAM (c) Walker3D NATGRAD (d) Walker3D TRPO

Figure 4.2: Performance of PICCOLO in various tasks. $x$ axis is iteration number and $y$ axis is sum of rewards. The curves are the median among 8 runs with different seeds, and the shaded regions account for $25\%$ percentile.

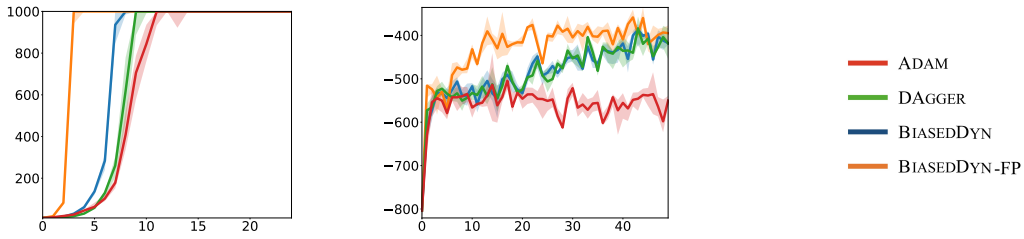
### 4.7.2 Nonlinear PICCOLO on IL Tasks

Next, we conduct an experiment on solving IL through the online learning approach described in Section 4.3.1. Two robot control tasks (CartPole and Reacher3D) powered by the DART physics engine [98] were used as the task environments.

**Expert and learner policies** To simulate the online IL task, we consider a neural network expert policy (with one hidden layer of $64$ units and $\tanh$ activation for the CartPole task and two hidden layers of $64$ units and $\tanh$ activation for the Reacher3D task). The expert policy is trained with additional Gaussian noise (with zero mean and a learnable variance) on the actions using a model-free policy gradient method (ADAM [95] with GAE [101]). While sharing the same architecture between the expert policy and the learner policy is not required in IL, here we adopted this constraint to remove the bias due to the mismatch between policy class and the expert policy to clarify the experimental results.

**Online IL setup** We choose FTRL the base algorithm as in DAGGER. The online loss in each round is $l_n(\pi) = \mathbb{E}_{s\sim d_{\pi_n}} D_{\mathrm{KL}}(\pi_s||\pi^\star_s)$, where $D_{\mathrm{KL}}$ denotes the KL divergence between two probability distributions (We observed that using $D_{\mathrm{KL}}(\pi_s||\pi^*_s)$ converges noticeably faster than using $D_{\mathrm{KL}}(\pi^*_s||\pi_s)$). The FTRL algorithm chosen is an non-adaptive algorithm: FTRL with constant learning rate (section 3.1 [96]). This leads to an update rule for PICCOLO: $\pi_n = \arg\min_{\pi\in\Pi} \hat{l}_n(\pi) + \sum_{m=1}^{n-1} l_m(\pi) + \frac{1}{\eta}\|\pi\|_2^2$, where the $\ell_2$ regularization is posed on the weights of the neural network. To generate the prediction function $\hat{l}_n$, we rely on biased dynamics models that are realized by perturbing the underlying physics parameters (mass and damping of revolute joints) of the systems by $20\%$. Furthermore, to investigate the performance gap between linear and nonlinear loss functions, we also run IL with ADAM as the base algorithm.

**Simulation Results** We compare the results of the base algorithm (DAGGER), its PICCOLO'ed version (BIASEDDYN and BIASEDDYN-FP), and ADAM that uses linear losses,




(a) CartPole (b) Reacher3D (c) legend

Figure 4.3: Performance comparison of ADAM, DAGGER, and PICCOLO'ed DAGGER with a biased dynamics model. $x$ axis is iteration number and $y$ axis is sum of rewards (the greater the better). The curves are the median among 4 runs with different seeds, and the shaded regions account for $25\%$ percentile. BIASEDDYN indicates that the mass of each individual robot link and damping of revolute joint is either increased or decreased by $80\%$ with probability 0.2 respectively. And BIASEDDYN-FP is approximated by applying three update steps (4.10) by gather a new batch of data using dynamics model in each round.

in terms of policy convergence rate. In Figure 4.3, the gap between the DAGGER and ADAM curves indicate that there's a performance gap between using linear and nonlinear loss functions. This agrees with our previous intuition in Section 4.6 that although online learning with nonlinear loss function has no obvious advantage in terms of regret bound, nonlinear loss functions capture more information and can potentially lead to faster policy improvement. This implies an interesting interplay between computation and sample efficiency in policy learning, which desires further investigation. Moreover, Figure 4.3 shows that PICCOLO can effectively leverage the useful information in an inaccurate dynamics model to accelerate policy learning. And the benefit becomes even more significant under the fixed-point formulation (4.8).

## 4.8 Conclusion

PICCOLO is a general reduction-based framework for solving predictable online learning problems. It can be viewed as an automatic strategy for generating new algorithms that can leverage prediction to accelerate convergence. Furthermore, PICCOLO uses the Correction Step to recover from the mistake made in the Prediction Step, so the presence of modeling errors does not bias convergence, as we show in both the theory and experiments. The

design of PICCOLO leaves open the question of how to design good predictive models. While PICCOLO is robust against modeling error, the accuracy of a predictive model can affect its effectiveness. PICCOLO only improves the performance when the model can make non-trivial predictions. In the experiments, we found that off-policy and simulated gradients are often useful, but they are not perfect. It would be interesting to see whether a predictive model that is trained to directly minimize the prediction error can further help policy learning. Finally, we note that, despite the focus of this chapter is on policy optimization, PICCOLO can naturally be applied to other optimization and learning problems.

# CHAPTER 5

# EXPLAINING FAST IMPROVEMENT IN ONLINE POLICY OPTIMIZATION

In Chapter 4, we developed a general reduction-based framework for solving predictable online learning problems. The framework is an automatic strategy for generating new online algorithms that can leverage *prediction* to accelerate convergence while being robust to modelling error. Based on the paradigm pioneered by Ross et al. [78] that views policy optimization as an online learning problem, our framework for accelerating online algorithms is directly applicable for solving policy optimization problems. Furthermore, effective and non-trivial predictions can be synthesized using dynamics models.

In this chapter, we provide new theoretical foundations of the online policy optimization (OPO) paradigm which therefore further justify the effectiveness of our predictable online learning framework towards solving policy optimization: we provide an explanation of the fast policy improvement phenomenon observed in practice that's usually much faster than existing theory suggests. Concretely, let $\epsilon$ denote the policy class bias and assume the online loss functions are convex, smooth, and non-negative. We prove that, after $N$ rounds of OPO with stochastic feedback, the policy converges in $\tilde{O}(1/N + \sqrt{\epsilon/N})$ in both expectation and high probability. In other words, we show that adopting a sufficiently expressive policy class in OPO has two benefits: both the convergence rate increases and the performance bias decreases, as the policy class becomes reasonably rich. This new theoretical insight is further verified in an online imitation learning experiment.

## 5.1 Introduction

Viewing policy optimization as no-regret online learning [102] has recently gained traction in the machine learning community. Pioneered by Ross et al. [78], this reduction was designed to mitigate the compounding error resulting from multi-step action executions in sequential

decision problems. Since this work was published, significant progress has been made in both theory and practice: Online imitation learning (IL) has been validated on physical robot control tasks [103, 7], and high-performance algorithms have been developed for structured prediction [79, 104, 105]. Similar ideas have also been applied in reinforcement learning [8] and system identification [106, 107].

Here we collectively call these algorithms *online policy optimization* (OPO) in order to capture the breadth of this framework. The main idea of OPO is to treat each policy in the sequential decision problem as an online decision* in online learning: the designer defines a sequence of online losses, such that the regret rate in the online learning problem implies the *speed* of policy improvement, and the minimal loss witnessed by the policy class determines the policy performance *bias* in the sequential decision making problem. When this loss qualification holds and the aforementioned performance bias is small, running a no-regret online algorithm in this online learning problem can generate policies with performance guarantees.

Because the online losses in OPO are designed to satisfy the performance relationship with respect to the given sequential decision making problem, the resulting online learning problem has a mixture of different properties, such as predictability, continuity, and stochasticity [108]. The interactions of these properties make the classic adversary-style online learning analysis taken by Ross et al.[78] overly conservative, creating a mismatch between provable theoretical guarantees and the learning phenomena observed in practice. This reality gap has motivated researchers to study deeper the theoretical underpinnings of OPO [80, 9, 109].

In this work, we are interested in explaining the fast policy improvement of OPO observed in practice, which existing OPO theory fails to capture. When the online loss functions are convex and Lipschitz, typical analyses of regret and martingale concentration [78, 110] suggest an on-average convergence rate in $O(1/\sqrt{N})$ after $N$ rounds. However, empir-

*The online decision in the iterative process of online learning should not be confused with the decisions made at each time step in sequential decision making.

ically, OPO algorithms learn much faster; for example, the online IL algorithm DAgger [78] learned to mimic a model predictive control (MPC) policy for autonomous off-road driving in only three rounds in [7]. Although the convergence rate improves to $\tilde{O}(1/N)$ when the online losses are strongly convex [80], this condition can be difficult to satisfy especially when the policy class is large, such as a linear function class built on high-dimensional features. The empirical effectiveness and sample efficiency of OPO demand alternative explanations.

We prove a new problem-dependent convergence rate for OPO that is adaptive to the performance bias from using a limited policy class. Interestingly, we show that an OPO algorithm can learn faster as this performance bias becomes smaller. In other words, adopting a sufficiently expressive policy class in OPO has two benefits: as the policy class becomes reasonably rich, both the learning speed increases and the performance bias decreases. Concretely, let $\epsilon$ denote the policy class bias. Under the assumptions that the online losses are convex, smooth, and non-negative, we give a convergence rate in $\tilde{O}(1/N + \sqrt{\epsilon/N})$ both in expectation and in high probability for OPO algorithms using stochastic feedback. This new result shows a transition from the faster rate of $\tilde{O}(1/N)$ to the usual rate of $\tilde{O}(1/\sqrt{N})$ as the policy class bias increases.

This type of problem-dependent convergence rate has been studied in various learning settings. When the loss functions are convex, smooth, and non-negative, similar rates have been shown in statistical learning [111], stochastic optimization [112, 113], and online learning [111]. Inspired by these results, we derive the bias-dependent improvement rate for OPO. The mix of non-stationarity and stochasticity in the online loss functions of OPO makes the analysis here challenging; indeed, previous analyses tackle only one of these two properties. Our fast high-probability bound is made possible by resorting to a recent martingale concentration result that depends only on path-wise statistics [114].

We conclude by corroborating the new theoretical findings with experimental results of online IL. The detailed proofs for this chapter can be found in Appendix C.

## 5.2 Background: Online Policy Optimization

### 5.2.1 Policy Optimization

We consider the policy optimization problem defined in Chapter 4 (Section 4.2), which we review here briefly. Policy optimization aims to find a high-performance policy in a policy class $\Pi$ for sequential decision making problems. Typically, it models the world as a Markov decision process (MDP), defined by an initial state distribution, transition dynamics, and an instantaneous state-action cost function [115]. This MDP is often assumed to be unknown to the learning agent; therefore the learning algorithm for policy optimization needs to perform systematic exploration in order to discover good policies in $\Pi$. Concretely, let us consider a policy class $\Pi$ that has a one-to-one mapping to a parameter space $\Theta$, and let $\pi_\theta$ denote the policy associated with the parameter $\theta \in \Theta$. That is, $\Pi = \{\pi_\theta : \theta \in \Theta\}$. The goal of policy optimization is to find a policy $\pi_\theta \in \Pi$ that minimizes the expected cost,

$$J(\pi) := \mathbb{E}_{s\sim d_{\pi_\theta}} \mathbb{E}_{a\sim\pi_\theta}[c(s,a)], \tag{5.1}$$

where $s$ and $a$ are the state and the action, respectively, $c$ is the instantaneous cost function and $d_{\pi_\theta}$ denotes the average state distribution over the problem horizon induced by executing policy $\pi_\theta$ starting from a state sampled from the initial state distribution. The problem formulation in (5.1) applies to various settings on problem horizon and discount rate, where the main difference is how the average state distribution is defined; e.g., for a discounted problem, $d_{\pi_\theta}$ is defined by a geometric mean, whereas $d_{\pi_\theta}$ corresponds to the stationary state distribution for average infinite-horizon problems.

### 5.2.2 Reducing Policy Optimization to Online Learning

OPO works by devising a sequence of online loss functions $l_n$ such that *no regret* and *small policy class bias* imply good policy performance in the original sequential decision

**Algorithm 4** Online Policy Optimization (OPO)

**Input:** Initial policy $\pi_{\theta_1}$, and an online algorithm $\mathcal{A}$
**Output:** The best policy in the sequence of policies $\{\pi_{\theta_n}\}_{n=1}^{N}$

```
Initialize A with the initial policy π_{θ_1}
for n from 1 to N do
    Define the online loss function l_n based on π_{θ_n}
    Execute policy π_{θ_n} in the MDP to gather samples
    Approximate l_n using an unbiased sample-based estimate l̂_n, which satisfies E[l̂_n] = l_n
    Pass l̂_n to A and use the return of A to update policy to π_{θ_{n+1}}
end for
Set π̄_N = π_{K−1}
```

problem. Below we use online imitation learning (IL) as a concrete example of OPO to illustrate this idea.

**Illustrative Example** Online IL performs policy optimization using an interactive expert policy $\pi_{\mathrm{e}}$. Instead of minimizing (5.1) directly, online IL minimizes an upper bound of the performance difference between the policy $\pi_\theta$ and the expert $\pi_{\mathrm{e}}$:

$$J(\pi) - J(\pi_{\mathrm{e}}) \leq O\left(\mathbb{E}_{s\sim d_{\pi_\theta}} \mathbb{E}_{a\sim\pi_\theta}[D_{\pi_{\mathrm{e}}}(s,a)]\right). \tag{5.2}$$

The loss $D_{\pi_{\mathrm{e}}}(s,a)$ represents whether an action $a$ is similar to the action taken by expert policy $\pi_{\mathrm{e}}$ at state $s$ [78, 79, 105, 9], which can be constructed as statistical distances (e.g., Wasserstein distance and KL divergence) or their upper bounds. Although the surrogate function on the right-hand side of (5.2) resembles (5.1), it has an additional nice property [80]: if the policy class $\Pi$ has enough capacity to contain the expert policy $\pi_{\mathrm{e}}$, then there is a policy $\pi_\theta \in \Pi$ such that, for *all* the states,

$$\mathbb{E}_{a\sim\pi_\theta}[D_{\pi_{\mathrm{e}}}(s,a)] = 0. \tag{5.3}$$

Leveraging the realizability property in (5.3), online IL minimizes the surrogate function in (5.2) by solving an online learning problem: Let parametric space $\Theta$ be the decision set (i.e.,

policies) in online learning; we can define the online loss in round $n$ as

$$l_n(\theta) = \mathbb{E}_{s\sim d_{\pi_{\theta_n}}} \mathbb{E}_{a\sim\pi_\theta}[D_{\pi_\mathrm{e}}(s,a)], \tag{5.4}$$

where $\theta_n \in \Theta$ is the online decision made by the learning algorithm in round $n$. The main benefit of this indirect approach is that, e.g., the sampled gradient of $l_n(\theta)$ in (5.4) is less noisy than that of the surrogate problem in (5.2), because the average state distribution $d_{\pi_{\theta_n}}$ in (5.4) is not considered as a function of the policy parameter $\theta$. The influence of the policy parameter on the change in the average state distribution can be ignored here, because of the realizable property in (5.3). When the expert policy $\pi_\mathrm{e}$ is only nearly realizable by the policy class $\Pi$ (that is, (5.3) can only be satisfied up to a certain error), optimizing the policy with this online learning reduction would suffer from an extra performance bias due to using a limited policy class, as we will later discuss in Section 5.2.3.

**General Learning Protocol** OPO algorithms in general closely follow the design idea we showed in the online IL example. First, an online loss is selected for the specific domain to satisfy conditions similar to (5.2) and (5.3) (or their approximations). An online algorithm $\mathcal{A}$ is then selected to optimize the policy with respect to the online loss functions. As a summary, Algorithm 4 illustrates the iterative process of a general OPO algorithm, where we take into account that in practice the MDP is unknown and therefore the online loss $l_n$ needs to be further approximated by finite samples as $\hat{l}_n$. At the end, the online algorithm $\mathcal{A}$ would generate a sequence of policies $\{\pi_{\theta_n}\}_{n=1}^N$. By this reduction, performance guarantees can be obtained for the best policy in this sequence, as we will show next.

### 5.2.3 Performance Guarantees in Online Policy Optimization

Now that we have reviewed the algorithmic aspects of OPO, we provide a brief tutorial of the theoretical foundation of OPO and the known convergence results that show exactly how regret and policy class bias are related to the performance in the original sequential

decision problem. To this end, let us formally define the regret and the policy class bias: For a sequence of online loss functions $\{f_n\}_{n=1}^N$ and decisions $\{\theta_n\}_{n=1}^N$, the regret in online learning is defined as

$$\text{Regret}(f_n) = \sum f_n(\theta_n) - \min_{\theta\in\Theta} \sum f_n(\theta). \tag{5.5}$$

Note that, for brevity, the range in $\sum_{n=1}^N$ is omitted in (5.5) and we will continue to do so in the following as long as the range is clear from the context. In addition to the regret, we introduce two problem-dependent biases of the decision set $\Theta$ (the equivalence of the policy class $\Pi$).

**Definition 2.** For the sampled loss functions $\hat{l}_n$ experienced by running Algorithm 4, we define $\hat{\epsilon} = \frac{1}{N}\min_{\theta\in\Theta}\sum \hat{l}_n(\theta)$ and $\epsilon = \frac{1}{N}\min_{\theta\in\Theta}\sum l_n(\theta)$, where $l_n(\theta) = \mathbb{E}[\hat{l}_n(\theta)]$ for a given $\theta$.

A typical OPO analysis uses the regret and the policy class biases $\epsilon$ and $\hat{\epsilon}$ to decompose the cumulative loss $\sum l_n(\theta_n)$ to provide policy performance guarantees. Specifically, define $\theta^\star \in \arg\min_{\theta\in\Theta}\sum l_n(\theta)$. By (5.5) and Definition 2, we can write

$$\begin{aligned}\sum l_n(\theta_n) &= \text{Regret}(\hat{l}_n) + \Big(\sum l_n(\theta_n) - \hat{l}_n(\theta_n)\Big) + N\hat{\epsilon} &(5.6)\\ &\le \text{Regret}(\hat{l}_n) + \Big(\sum l_n(\theta_n) - \hat{l}_n(\theta_n)\Big) + \Big(\sum \hat{l}_n(\theta^\star) - l_n(\theta^\star)\Big) + N\epsilon &(5.7)\end{aligned}$$

where, in both (5.6) and (5.7), the first term is the online learning regret, the middle term(s) are the generalization error(s), and the last term is the policy class bias. Because $l_n(\theta_n)$ in OPO provides an upper bound on the policy performance (see (5.2)), making the cumulative loss $\sum l_n(\theta_n)$ small implies performance guarantees on the best policy in the sequence $\{\pi_{\theta_n}\}_{n=1}^N$. For simplicity, hereafter, we will abstract away the domain details of the sequential decision problem and focus on the size of $\sum l_n(\theta_n)$, as it directly implies the policy performance.

In a nutshell, existing convergence results of OPO are applications of (5.6) and (5.7) with different upper bounds on the regret and the generalization errors [78, 106, 79, 105]. For example, when the sampled loss functions $\{\hat{l}_n\}$ are bounded, the generalization error(s) (i.e., the middle term(s) in (5.6) and (5.7)) can be bounded by $\tilde{O}(\sqrt{N})$ with high probability by Azuma's inequality (see [110] or Chapter 9 in [84]). Together with an $O(\sqrt{N})$ bound on the regret (which is standard for online convex losses) [84, 96], it implies that the average performance $\frac{1}{N}\sum l_n(\theta_n)$ and the best performance $\min_n l_n(\theta_n)$ converge to $\hat{\epsilon}$ or $\epsilon$ at the speed of $\tilde{O}(1/\sqrt{N})$.

However, the rate above does not fully justify the fast improvement of OPO observed in practice [116, 105, 7, 86], as we will also show experimentally in Section 5.5. While faster rates in $\tilde{O}(1/N)$ can be obtained for strongly convex loss functions [78, 80], these results are not very assuring either: the strong convexity assumption does not usually hold, especially when an expressive policy class $\Pi$ is used to reduce the policy class bias. Thus, alternative explanations are needed.

## 5.3 Bias-Dependent Convergence Rates

In this section, we present new policy convergence rates that are adaptive to the performance biases in Definition 2. The full proof of these theorems are provided in the Appendix C.

### 5.3.1 Setup and Assumptions

We suppose the parameter space of the policy class $\Theta$ is a closed convex subset of a Hilbert space $\mathcal{H}$ that is equipped with norm $\|\cdot\|$. Note that $\|\cdot\|$ is not necessarily the norm induced by the inner product. We will denote its dual norm as $\|\cdot\|_*$, which is defined as $\|x\|_* = \max_{\|y\|=1}\langle x, y\rangle$.

We define admissible algorithms to broaden the scope of OPO algorithms that our analysis covers.

**Definition 3.** We say an online algorithm $\mathcal{A}$ is *admissible* if there exists $R_{\mathcal{A}} \in [0, \infty)$ such that given any $\eta > 0$ and any sequence of differentiable convex functions $\{f_n\}$, $\mathcal{A}$ can achieve $\mathrm{Regret}(f_n) \leq \mathrm{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle) \leq \frac{1}{\eta} R_{\mathcal{A}}^2 + \frac{\eta}{2} \sum \|\nabla \hat{l}_n(\theta_n)\|_*^2$, where $\theta_n$ is the decision made by $\mathcal{A}$ in round $n$.

We will assume that Algorithm 4 is realized by an admissible online learning algorithm $\mathcal{A}$. This assumption is satisfied by common online algorithms, such as mirror descent [117] and Follow-The-Regularized-Leader [96] under first-order or full-information feedback, where $\eta$ in Definition 3 corresponds to a constant stepsize that's chosen before seeing the online losses, and $R_{\mathcal{A}}$ measures that size of the decision set $\Theta$.

Finally, we formally define convex, smooth, and non-negative (CSN) functions; we will assume the online loss $l_n$ in OPO and its sampled version $\hat{l}_n$ belong to this class.

**Definition 4.** A function $f : \mathcal{H} \to \mathbb{R}$ is *CSN* if $f$ is convex, $\beta$-smooth, and non-negative.

Several popular loss functions used in OPO (e.g., squared $\ell_2$-loss and KL-divergence) are indeed CSN (Definition 4) (see Section 5.4 for examples). If the losses are not smooth, several smoothing techniques in the optimization literature are available to smooth the losses locally, e.g., Nesterov's smoothing [118], Moreau-Yosida regularization [119], and randomized smoothing [120].

### 5.3.2 Convergence Rate in Expectation

Our first contribution is a bias-dependent convergence rate in expectation.

**Theorem 3.** *In Algorithm 4, suppose $\{\hat{l}_n\}$ is CSN and the online algorithm $\mathcal{A}$ is admissible. Let $\hat{\epsilon} = \frac{1}{N} \min_{\theta \in \Theta} \sum \hat{l}_n(\theta)$ be the bias, and let $\hat{E}$ be such that $\hat{E} \geq \hat{\epsilon}$ almost surely. Choose $\eta$ for $\mathcal{A}$ to be $\frac{1}{2\left(\beta + \sqrt{\beta^2 + \frac{\beta N \hat{E}}{2R_{\mathcal{A}}^2}}\right)}$. Then in expectation*

$$\frac{1}{N} \sum l_n(\theta_n) - \hat{\epsilon} \leq \frac{8\beta R_{\mathcal{A}}^2}{N} + \sqrt{\frac{8\beta R_{\mathcal{A}}^2 \hat{E}}{N}} \tag{5.8}$$

*Proof Sketch.* The rate (5.8) follows from analyzing the regret and the generalization error in the decomposition in (5.6). First, under the assumption of CSN loss functions and admissible online algorithms, the online regret can be bounded by an extension of the bias-dependent regret bound that is stated for mirror descent in [111, Theorem 2], whose average gives the rate in (5.8). Second, the generalization error in (5.6) vanishes in expectation because it is a martingale difference sequence. □

The rate in (5.8) suggests that an OPO algorithm can learn faster as the policy class bias becomes smaller; this is reflected in the transition from the usual rate $O(1/\sqrt{N})$ to the faster rate $O(1/N)$ when the bias goes to zero. Notably, the rate in (5.8) does not depend on the dimensionality of $\mathcal{H}$ but only on $R$, which one can roughly think of as the largest norm in $\Theta$. Therefore, we can increase the dimension of the policy class to reduce the bias (e.g.,by using reproducing kernels [121]) as long as the diameter of $\Theta$ measured by norm $\|\cdot\|$ stays controlled.

### 5.3.3 Convergence Rate in High Probability

We show that a similar bias-dependent convergence rate to (5.8) also holds with high probability.

**Theorem 4.** *Under the same assumptions and setup of Theorem 3, further assume that there is $G \in [0, \infty)$ such that, for any $\theta \in \Theta$, $\|\nabla \hat{l}_n(\theta)\|_* \leq G$. For $\delta < 1/e$, with probability at least $1 - \delta$,*

$$\frac{1}{N}\sum l_n(\theta_n) - \epsilon = O\left(\frac{C\beta R^2}{N} + \sqrt{\frac{C\beta R^2(\hat{E}+\epsilon)}{N}}\right) \tag{5.9}$$

*where $R_\Theta = \max_{\theta\in\Theta} \|\theta\|, R = \max(1, R_\Theta, R_{\mathcal{A}}), C = \log(1/\delta)\log(GRN)$.*

We remark that the uniform bound $G$ on the norm of the gradients only appears in logarithmic terms. Therefore, this rate stays reasonable when the loss functions have gradients whose norm grows with the size of $\Theta$, such as the popular the squared loss.

To prove Theorem 4, one may attempt to apply basic martingale concentration properties on the martingale difference sequences (MDSs) in (5.6) and (5.7), as in the proof of Theorem 3. However, taking this direct approach will bring back the slow rate of $O(1/\sqrt{N})$. To the best of our knowledge, sharp concentration inequalities for the counterparts of MDS in other learning settings cannot be adapted here in a straightforward way: for example, [111] relies on an argument on local Rademacher complexities (which does not have obvious extension to nonstationary losses) and [122] assumes that the losses are both Lipschitz and strongly convex (our goal is to relax these assumptions).

### 5.3.4 Proof Sketch for Theorem 2

The key to avoid the above slow rate due to the direct application of martingale concentration analyses on the MDSs in (5.6) and (5.7) is to take a different decomposition of the cumulative loss. Here we construct two *new* MDSs in terms of the gradients: recall $\epsilon = \min_{\theta\in\Theta} \sum l_n(\theta)$ and let $\theta^\star = \arg\min_{\theta\in\Theta} \sum l_n(\theta)$. Then by convexity of $l_n$, we can derive

$$\begin{aligned}
&\sum l_n(\theta_n) - N\epsilon \\
\leq &\sum \underbrace{\langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n), \theta_n \rangle}_{\text{MDS}} - \sum \underbrace{\langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)}_{\text{MDS}}, \theta^\star \rangle + \text{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle)
\end{aligned} \tag{5.10}$$

Our proof is based on analyzing these three terms. For the MDSs in (5.10), we notice that, for smooth and non-negative functions, the squared norm of the gradients can be bounded by its function value.

**Lemma 4** (Lemma 3.1 [111])**.** *Suppose a function $f : \mathcal{H} \to \mathbb{R}$ is $\beta$-smooth and non-negative, then for any $x \in \mathcal{H}$, $\|\nabla f(x)\|_*^2 \leq 4\beta f(x)$.*

Lemma 4 enables us to properly control the second-order statistics of the MDSs in (5.10). By a recent vector-valued martingale concentration inequality that depends only

on second-order statistics [114], we obtain a self-bounding property for (5.10) to get fast concentration rate.

Besides analyzing the MDSs, we need to bound the regret to the linear functions defined by the gradients (the last term in (5.10)). Since this last term is linear, not CSN, the bias-dependent online regret bound in the proof of Theorem 3 does not apply. Nonetheless, because these linear functions are based on the gradients of CSN functions, we discover that their regret rate actually obeys the exact same rate as the regret to the CSN loss functions. This is notable because the regret to these linear functions upper bounds the regret to the CSN loss functions.

Combining the bounds on the MDSs and the regret, we obtain the rate in (5.9).

## 5.4 Case Studies

In this section, we use two concrete applications of OPO to show how the new theoretical results in Section 5.3 improve existing understanding of the policy improvement speed.

### 5.4.1 Online Imitation Learning

Online IL [78] has demonstrated successes in solving many real-world sequential decision making problems [116, 123, 7]. When the action space is discrete, a popular design choice is to set $D_{\pi_e}(s, a)$ in (5.4) as the hinge loss [78] (i.e.,the loss function used in SVM [121]). For continuous domains, $\ell_1$-loss becomes a natural alternative for defining $D_{\pi_e}(s, a)$, which, e.g., is adopted for high-speed autonomous off-road driving [7]. When the policy is linear in the parameters, one can verify that these loss functions are convex and non-negative, though not strongly convex. Therefore, existing convergence results can only guarantee an $O(1/\sqrt{N})$ policy improvement rate, which does not reflect the fast convergence observed in the experiments [78, 7].

Although our new theorems are not directly applicable to these non-smooth loss functions, we can apply our new results to a smoothed version of these non-negative convex loss

functions. For instance, applying the Huber approximation (an instantiation of Nesterov's smoothing) [118] to "smooth the tip" of these $\ell_1$-like losses yields a globally smooth function with respect to the $\ell_2$-norm. Because the smoothing mainly changes where the loss is close to zero, our new theorems suggest that, when the policy class is expressive enough, learning with these $\ell_1$-like losses would converge in a $\tilde{O}(1/N)$ rate before the policy gets very close to the expert policy during policy optimization.

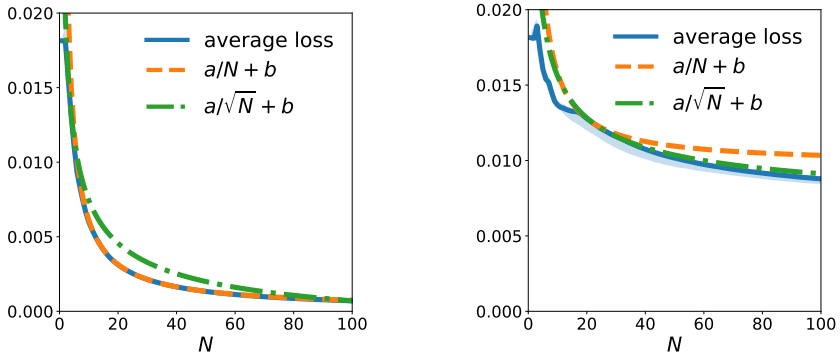
### 5.4.2 Interactive System Identification for Model-based RL

Interactive system identification (ID) is a technique that interleaves data collection and dynamics model learning for robust model-based RL. Ross et al. [106] show that interactive system ID can be analyzed under the OPO framework, where the regret guarantee implies learning a dynamics model that mitigates the train-test distribution shift problem [124, 106]. Let $T$ and $T_\theta$ denote the true and the learned dynamics, respectively. A common online loss for interactive system ID is $l_n(\theta) = \mathbb{E}_{(s,a)\sim\frac{1}{2}d_{T_{\theta_n}}+\frac{1}{2}\nu}\left[D_{s,a}(T_\theta||T)\right]$, where $D_{s,a}(T_\theta||T)$ is some distance between $T$ and $T_\theta$ under state $s$ and action $a$, $\nu$ is the state-action distribution of an exploration policy, and $d_{T_{\theta_n}}$ is the state-action distribution induced by running an optimal policy with respect to the model $T_{\theta_n}$. When the model class is expressive enough to contain the $T$, it holds $l_n(\theta) = 0$ for some $\theta \in \Theta$ (cf. (5.3)).

Suppose that the states and actions are continuous. A common choice for $D_{s,a}(T_\theta||T)$ in learning deterministic dynamics is the squared error $D_{s,a}(T_\theta||T) = \|T_\theta(s,a) - s'\|_2^2$ [106], where the $s'$ is the next state in the true transition of $T$. If $T_\theta$ is linear in $\theta$ or belongs to a reproducing kernel Hilbert space [121], the sampled loss function $\hat{l}_n$ is CSN. Alternatively, when learning a probabilistic model, $D_{s,a}$ can be selected as the KL-divergence [106]; it is known that if $T_\theta$ belongs to the exponential family of distributions, the KL divergence, and hence $\hat{l}_n$, are smooth and convex [125]. If the sample size is large enough, $\hat{l}_n$ becomes non-negative in high probability.

As these online losses are CSN, our theoretical results apply and suggest a convergence

rate in $\tilde{O}(1/N)$. On the contrary, the finite sample analysis conducted in [106] uses the standard online-to-batch techniques [110] and can only give a rate of $O(1/\sqrt{N})$. Our new results provide a better explanation to justify the fast policy improvement speed observed empirically, e.g., Figure 2 of [106].




(a) unbiased policy class (b) biased policy class

Figure 5.1: The convergence rates of online IL when the policy class has zero bias (Figure 5.1a) and an additional bias due to an $\ell_2$-norm constraint on the weights (Figure 5.1b). The rates are obtained by fitting the curve of the average loss $\frac{1}{N}\sum_{n=1}^{N} l_n(\theta_n)$ with parametric $O(1/N)$ and $O(1/\sqrt{N})$ upper bounds to minimize $\ell_1$-error. The average loss curve is the median, and the shaded region represents $10\%$ and $90\%$ percentile, over 4 random seeds, due to the randomness in the initial state of the MDP and the initialization of the policy.

## 5.5 Experimental Results

Although the main focus of this chapter is the new theoretical insights, we conduct experiments to provide evidence that the fast policy improvement phenomena indeed exist, as our theory predicts. We verify the change of rates due to policy class capacity by running an online IL experiment in the CartPole balancing task in OpenAI Gym [97] with DART physics engine [98].

**MDP setup** The goal of the CartPole task is to keep the pole upright by controlling the acceleration of the cart. The start state is a configuration with a small uniformly sampled offset from being static and vertical, and the dynamics is deterministic. In each time step, if the pole is maintained within a threshold from being upright, the learner receives an

instantaneous reward of one; otherwise, the learner receives zero reward and the episode terminates. This MDP has a 4-dimensional continuous state space and a 1-dimensional continuous action space.

**Expert and learner policies** To simulate the online IL task, we consider a neural network expert policy (with one hidden layer of $64$ units and $\tanh$ activation). The expert policy is trained with additional Gaussian noise (with zero mean and a learnable variance) on the actions using a model-free policy gradient method (ADAM [95] with GAE [101]). We let the learner policy be another neural network that has exactly the same architecture as the expert policy; we copy the weights for the hidden layer from those of the expert policy and randomly initialized the weights of the output layer. During training, only the weights of the learner's output layer were updated. In this way, we can view the learner as a *linear* policy using the representation of the expert policy.

**Online IL setup** We emulate online IL with unbiased and biased policy classes. We define the unbiased class as all the policies satisfying the above architecture, whereas we define the biased policy class by imposing an additional $\ell_2$-norm constraint on the learner's weights in the second layer so that the learner cannot perfectly mimic the expert policy. We select $l_n(\theta) = \mathbb{E}_{s\sim d_{\pi_{\theta_n}}}[H_\mu(\pi_\theta(s) - \pi_{\mathrm{e}}(s))]$ as the online loss in IL (see Section 5.2.2), where $H_\mu$ is the Huber function defined as $H_\mu(x) = \frac{1}{2}x^2$ for $|x| \leq \mu$ and $\mu|x| - \frac{1}{2}\mu^2$ for $|x| > \mu$. In the experiments, $\mu$ is set to $0.05$; as a result, $H_\mu$ is linear when its function value is larger than $0.00125$. Because the learner's policy is linear, this online loss is CSN in the unknown weights of the learner. We use ADAM [95] to optimize the learner policy with constant stepsize $0.01$.

**Simulation results** We compare the results in the unbiased and the biased settings, in terms of how the average loss $\frac{1}{N}\sum_{n=1}^{N} l_n(\theta_n)$ changes as the number of rounds $N$ in online learning increases. To see whether the rate of the average loss is $O(1/N)$ or $O(1/\sqrt{N})$, we

Table 5.1: Comparison of different learning settings.

| Setting | Info. | Stochastic | Stationary | Estimator | Excess loss |
|---|---|---|---|---|---|
| OPO | $\hat{l}_n$ | Yes | No | online | $\sum l_n(\theta_n) - \min \sum l_n(\theta)$ |
| Online learning | $l_n$ | No | No | online | $\sum l_n(\theta_n) - \min \sum l_n(\theta)$ |
| Statistical learning | $\hat{l}$ | Yes | Yes | ERM | $l(\theta_{\mathrm{ERM}}) - \min l(\theta)$ |
| Online-to-batch | $\hat{l}$ | Yes | Yes | online | $\sum l(\theta_n) - N \min l(\theta)$ |
| Stochastic bandits | $\hat{l}(\theta_n)$ | Yes | Yes | online | $\sum l(\theta_n) - N \min l(\theta)$ |

fit the curves using the parametric functions $f_1(N) = a\frac{1}{N} + b$ and $f_2(N) = a\frac{1}{\sqrt{N}} + b$. The parameters $a, b$ are obtained by solving a constrained convex program that minimizes the $\ell_1$-loss with a constraint that the graphs of the parametric functions lie above the curves of the average loss. The experimental results are depicted in Figure 5.1. In the unbiased setting (Figure 5.1a), the curve fits well with the parametric function $f_1$. By contrast, in the biased setting (Figure 5.1b), the curve aligns better with the parametric function $f_2$.

## 5.6 Related Work and Discussion

In this chapter, we prove new expected and high-probability convergence rates that depend on the policy class capacity for OPO problems. Our results are closely related to the problem-dependent rates studied in several more typical learning settings. We summarize the relationship between our work and other related results in Table 5.1, where we compare different learning settings in terms of the information available to the learner, the properties of the loss functions, and the form of excess loss.

In statistical learning, there is a new trend studying how the generalization of the empirical risk minimizer (ERM) depends on the properties of loss functions. Rates dependent on the bias due to the hypothesis class are shown for Lipschitz loss functions [126], and smooth loss functions [111]. These results are extended to ERM in general stochastic optimization in [112, 113].

Another line of research on problem-dependent convergence rates focus on online learning with adversarial loss sequences. Although online learning does not impose the i.i.d.

assumption on loss functions, comparator-dependent rates on the regret can also be proved for smooth [111] and smooth plus log-concave [127] online loss functions.

Finally, the research on online-to-batch conversion [128] studies the generalization in statistical learning through analyzing online learning problems with loss functions that are i.i.d. sampled. Cesa-Bianchi et al.[110] establish a fundamental connection between the regret in such online learning problems and the generalization error in statistical learning. Cesa-Bianchi and Gentile [129] relate the cumulative loss in online learning to the generalization in statistical learning, showing that a faster rate of generalization can be achieved if the cumulative loss is small.

In comparison, OPO concerns loss functions that are *both* stochastic and online; that is, we can view statistical and online learning as special cases of OPO. The interactions between noises and non-stationarity make the analysis of OPO especially interesting. We tackle these challenges by joining analysis techniques from stochastic optimization [112] and online learning [111]. As a consequence, we are able to develop new insights to explain certain fast convergence phenomena of OPO, which existing OPO theory fails to capture.

However, our current results cannot explain all the fast improvements of OPO observed in practice. The analyses here are based on the assumption of using convex and smooth loss functions. This assumption would be violated, for example, with a deep neural network policy based on with ReLU activation; yet fast empirical convergence rates of these networks have been shown in OPO [7]. Nonetheless, we envision that the insights from this chapter can provide a promising starting point to better understanding the behaviors of OPO, and to suggest directions for designing new OPO algorithms that proactively leverage these self-bounding regret properties to achieve faster learning.

# CHAPTER 6

# TRAJECTORY-WISE CONTROL VARIATES FOR POLICY OPTIMIZATION

In Chapter 4 and Chapter 5, we considered policy optimization in the on-policy episodic setting, and besides providing new theoretical insight for fast policy improvement, we developed an algorithmic framework for designing methods that can leverage dynamics models or, more general, predictive models to accelerate learning while avoiding performance bias due to modeling errors.

In this chapter, we focus on another class of *bias-free* policy optimization algorithms that can also significantly benefit from dynamics models: control variate (CV) methods [130, 131, 132]. CVs use models to reduce the variance of sampled gradients in any *policy gradient* method to improve convergence, and they are orthogonal to the PICCOLO framework in Chapter 4. Interestingly, CV and PICCOLO can be naturally combined together: PICCOLO's performance depends on the accuracy of predicting the sampled gradient of next round before seeing it, and control variates reduce the variance of the sampled gradient after receiving it, making it easier to predict and thus improving bounds of PICCOLO.

CV) have been studied for decades to reduce the variance of policy gradient estimates without introducing bias. Examples include the early use of baselines, state dependent CVs, and the more recent state-action dependent CVs. In this chapter, we analyze the properties and drawbacks of previous CV techniques and, surprisingly, we find that these works have overlooked an important fact that Monte Carlo gradient estimates are generated by *trajectories* of states and actions. We show that ignoring the correlation across the trajectories can result in suboptimal variance reduction, and we propose a simple fix: a class of *trajectory-wise* CVs, that can further drive down the variance. The trajectory-wise CVs can be computed recursively and require only learning state-action value functions like the previous CVs for policy gradient. We further prove that the proposed trajectory-wise CVs

are optimal for variance reduction under reasonable assumptions.

## 6.1 Introduction

Policy gradient methods [133, 134, 99, 135, 100, 136] are a popular class of model-free policy optimization algorithms. They have many advantages, including simple update rules and convergence guarantees [134, 89, 86, 137]. However, basic policy gradient methods, like REINFORCE [133], are also notoriously sample inefficiency. This can be attributed, at least in part, to the high variance in Monte Carlo gradient estimates, which stems from both policy stochasticity necessary for exploration as well as stochastic environmental dynamics. The high variance is further exacerbated as the horizon becomes longer and the dimension becomes higher. If the variance of gradient estimates can be reduced, then the learning speed of policy gradient methods can be accelerated [138, 136].

Variance reduction has been studied since early work of policy gradient methods. For example, function approximators (critics) have been adopted to (partially) replace the Monte Carlo estimates, which reduces variance but at the expense of bias in the search direction [134, 139, 140, 141, 101, 142]. This bias-variance tradeoff can work well in practice, but can also diverge when not tuned carefully [101, 143, 137].

Another line of research uses the control variate (CV) method from statistics, which can reduce variance in Monte Carlo methods without introducing bias [134, 144, 145, 146, 147, 148, 131, 149, 150]. For policy gradient algorithms, specialized CV methods have been proposed to take advantage of structures inherent in policy optimization. Especially, the state dependent CVs (also known as baselines or reward reshaping [144, 146]) have been thoroughly investigated [145]. Common state dependent CVs are constructed as approximators of the policy's value function, which admits update rules based on policy evaluation techniques. Overall, state dependent CVs are simple to implement and have been found to be quite effective, but they can still lead to detrimentally high variance, especially in problems that has a long horizon. This issue has motivated a recent development of

state-action dependent CVs [147, 148, 131, 149, 151] that can further reduce the variance due to randomness in the *actions* the previous state-only CVs fails to manage.

Considering the decades-long development of CV methods, one might wonder if there is a need for new policy gradient CV techniques. In this chapter, we argue that the past development of CVs for policy gradients has overlooked an important fact that the Monte Carlo gradient estimates are generated by rolling out a policy and collecting statistics along a *trajectory* of states and actions. Instead the focus has been on sampling *pairs* of states and actions, ignoring the correlation between states and actions *across* time steps. Recently Tucker et al. [149] empirically analyzed the variance of instantaneous state-action pairs and compared this to the variance correlations across time steps in multiple simulated robot locomotion tasks. They found that the variance due to long-term trajectories is often larger than the variance due to instantaneous state-action pairs. This finding implies that there is potential room for improvement.

We theoretically analyze the properties of previous CVs, and show that indeed the variance due to long-term trajectories can have non-negligible effects. Motivated by this observation, we propose a family of trajectory-wise CVs, called *TrajCV*, which recursively augments existing CVs with extra terms to *additionally* cancel this long-term variance. We show that TrajCV is particularly effective when the transition dynamics, despite unknown, is close to deterministic. Like existing CVs, TrajCV requires only approximates of the state-action value function (i.e., Q-function) of a policy, which can be benefit significantly from accurate dynamics models. Moreover, we prove that TrajCV is optimal for variance reduction under reasonable assumptions. These theoretical insights are validated in simulation.

## 6.2 Problem Setup and Background

We consider episodic policy optimization as described in Chapter 4. For easy exposition, we consider a finite-horizon Markov Decision Process (MDP) [152, 153] with horizon $T$, state space $\mathbb{S}$, action space $\mathbb{A}$, instantaneous cost function $c : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, initial state distribution

$p_1$, and dynamics $\mathcal{P}$. Given a parameterized *stochastic* policy class $\Pi$ the goal is to search for a policy in $\Pi$ that achieves low accumulated costs averaged over trajectories

$$J(\pi) = \mathbb{E}[\sum_{t=1}^{T} c(s_t, a_t)], \quad s_1 \sim p_1, \quad a_t \sim \pi_{s_t}, \quad s_{t+1} \sim \mathcal{P}_{s_t,a_t} \tag{6.1}$$

where $\mathcal{P}_{s,a}$ denotes the distribution of the next state after applying action $a \in \mathbb{A}$ at state $s \in \mathbb{S}$, and $\pi_s$ denotes the distribution of action at state $s \in \mathbb{S}$. Note that $s_t$ and $a_t$, i.e.,state and action at step $t$, are random variables due to the randomness in dynamics and policy. For simplicity of writing, we embed the time information into the definition of state. Therefore $c(s_t, a_t)$ can represent non-stationary cost functions. The randomness in (6.1) consists of the randomness in the start state, policy, and dynamics. In this work, we focus on the case where the dynamics $\mathcal{P}$ and the start state distribution $p_1$ are unknown, but the instantaneous cost $c$ is known.

**Notation** We will use subscript $_{i..j}$ to denote the set of random variables, i.e., $x_{1..5} = \{x_1, \ldots, x_5\}$, and $_{i:j}$ to denote summation (i.e., $c_{1:T} = \sum_{t=1}^{T} c_t$). As we will be frequently manipulating conditional distributions, we adopt the subscript notation below to write conditional expectation and variance. For $\mathbb{E}_{x|y}[f(x, y)]$ of some function $f$, $x$ denotes the random variable where the expectation is defined and $y$ denotes the conditioned one. Furthermore, for $f(x_{1..N}, y)$, we use $\mathbb{E}_{|y}[f(x_{1..N}, y)]$ as a shorthand to denote taking the expectation over all other random variables (i.e., $x_{1..N}$) conditioned on $y$. This subscript notation also applies to variance, which is denoted as $\mathbb{V}$.

### 6.2.1 Policy Gradient Methods: Pros and Cons

The goal of this chapter is to improve the learning performance of policy gradient methods [133, 89, 134, 99, 135, 141, 100, 136]. These algorithms treat minimizing (6.1) as a first-order stochastic non-convex optimization problem, where noisy, unbiased gradient estimates of $J$ in (6.1) are used to inform policy search. The basic idea is to apply the

likelihood-ratio method to derive the gradient of (6.1). Let us define $\rho_t = \nabla \log \pi_{s_t}(a_t)$, where $\nabla$ is the derivative with respect to the policy parameters, and define $Q_\pi$ as the Q-function of $\pi$; that is, $Q_\pi(s_t, a_t) = \mathbb{E}[c_{t:T}]$. Define $g_t = \rho_t c_{t:T}$, and $g = g_{1:T}$. Then it follows [133]

$$\nabla J(\pi) = \mathbb{E}[\textstyle\sum_{t=1}^{T} \rho_t Q_\pi(s_t, a_t)] = \mathbb{E}\left[g\right], \tag{6.2}$$

(6.2) is an expectation over trajectories generated by running $\pi$. Thus, we can treat the random vector $g$ as an unbiased estimate of $\nabla J(\pi)$, which can be computed by executing the policy $\pi$ starting from initial distribution and then recording the statistics $g_t$, for $t \in \{1, \ldots, T\}$. This technique is known as the Monte Carlo estimate, which samples i.i.d. trajectories from the trajectory distribution in (6.1) to approximate the expectation.

The policy gradient methods (e.g., REINFORCE [133]) optimize policies based on gradient estimates constructed using the above idea. They have numerous advantages, such as straightforward update rules and convergence guarantee [134, 89, 86, 137]. But simply using the Monte Carlo estimate $g$ in policy optimization (i.e., the vanilla implementation of REINFORCE) can result in poor performance due to excessive variance [141, 101]. Therefore, while ideally one can apply standard first-order optimization algorithms, such as mirror descent [154], with the random estimate $g$ to optimize policies, this often is not viable in practice. They would require a tremendous amount of trajectories in order to attenuate the high variance, making learning sample inefficient.

The high variance of $g$ is due to the exploration difficulty: in the worst-case, the variance of $g$ can grow exponentially in the problem's horizon [155, 156], as it becomes harder for the policy to visit meaningful states and get useful update information. Intuitively we can then imagine that policy optimization progress can be extremely slow, when the gradient estimates are noisy. From an optimization perspective, variance is detrimental to the convergence rate in stochastic optimization. For example, the number of iterations for

mirror descent to converge to an $\epsilon$-approximate stationary point is $O((\mathrm{Tr}\,(\mathbb{V}[g])+1)/\epsilon^2)$, increasing as the problem becomes more noisy [138]. Therefore, if the variance of estimates of (6.2) can be reduced, the policy gradient methods can be accelerated.

### 6.2.2 Variance Reduction and Control Variate

A powerful technique for reducing the variance in the Monte Carlo estimates is the CV method [157, 158]. Leveraging correlation between random estimates, the CV method has formed the backbone of many state-of-the-art stochastic optimization algorithms [159, 160, 161], in particular, practical policy gradient methods [145] because of the high-variance issue of $g$ discussed in the previous section. Below, we review the basics of the CV method as well as previous CV techniques designed for reducing the variance of $g$. Without loss of generality, we suppose only one trajectory is sampled from the MDP to construct the estimate of (6.2) and study the variance of different single-sample estimates. We remind that the variance can be always further reduced, when more i.i.d. trajectories are sampled (i.e.,using mini batches).

### 6.2.3 Control Variate Method and Difference Estimator

Consider the problem of estimating the expectation $\mathbb{E}[x]$, where $x$ is a (possibly multivariate) random variable. The CV method [157, 158] is a technique for synthesizing unbiased estimates of $\mathbb{E}[x]$ that potentially have lower variance than the naive sample estimate $x$. It works as follows: Assume that we have access to another random variable $y$, called the *CV*, whose expectation $\mathbb{E}[y]$ is cheaper to estimate than $\mathbb{E}[x]$. Then we can devise this new estimate by a linear combination,

$$x - \Omega^\top (y - \mathbb{E}[y]), \tag{6.3}$$

where $\Omega$ is a properly-shaped matrix. Due to the linearity of expectation, the estimate in (6.3) is unbiased. Suppose $y$ is in the same dimension as $x$. One can show that the optimal $\Omega$ is $\Omega^\star = \frac{1}{2}\mathbb{V}[y]^{-1}\left(\mathbb{V}[x,y] + \mathbb{V}[y,x]\right)$. When data are too scarce to estimate $\Omega^\star$, $\Omega$ usually set as the identity, which often works well when $y$ is positively correlated with $x$. The resulting estimate $x - (y - \mathbb{E}[y])$ is known as the *difference estimator* [158] and has variance $\mathbb{V}[x-y]$, meaning that if $y$ is close to $x$ then the variance becomes smaller. In the following, we concentrate on the design of difference estimators; we note that designing a good $\Omega$ is an orthogonal research direction.

### 6.2.4 Common Control Variates for Policy Gradient Methods

The art to various CV methods lies in the design of the correlated random variable $y$. The choice is often domain-dependent, based on how $x$ is generated. When estimating the policy gradient in (6.2), many structures (e.g., the Markov property) can be leveraged to design CVs. We discuss properties of these designs below. Following previous works (e.g., [145, 149]), here we focus on the policy gradient component $g_t$ of $g$ given in (6.2) for simplicity of exposition.*

The most commonly used CVs for policy gradient [133, 144, 145] are state-dependent functions $\hat{V} : \mathbb{S} \to \mathbb{R}$, which leads to the difference estimator

$$\hat{g}_t^{\mathrm{S}} = g_t - \left(\rho_t \hat{v}_t - \mathbb{E}_{a_t|s_t}[\rho_t \hat{v}_t]\right) = g_t - \rho_t \hat{v}_t, \quad \text{where } \hat{v}_t = \hat{v}(s_t), \tag{6.4}$$

and the expectation vanishes as $\mathbb{E}_{a_t|s_t}[\rho_t \hat{v}_t] = \hat{v}_t \nabla \mathbb{E}_{a_t|s_t}[1] = 0$.† Recently, *state-action CVs* $\hat{Q} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ have also been proposed [147, 148, 131, 149, 151, 162], in an attempt to reduce more variance through CVs that better correlate with $g_t$. The state-action CVs yields

*Without any assumption of the MDP, the variance of $g$ can be bounded by the variance of $g_t$ ( Appendix D.1.3). Tighter bounds can be derived when assumptions on the MDP is made, e.g., faster mixing rate [145].

†State dependent functions naturally include non-stationary constant baselines in our notation.

the difference estimator

$$\hat{g}_t^{\mathrm{SA}} = g_t - \left(\rho_t \hat{q}_t - \mathbb{E}_{a_t|s_t}[\rho_t \hat{q}_t]\right), \quad \text{where } \hat{q}_t = \hat{Q}(s_t, a_t). \tag{6.5}$$

Usually $\hat{V}$ and $\hat{Q}$ are constructed as function approximators of the value function $V_\pi$ and the Q-function $Q_\pi$ of the current policy $\pi$, respectively, and learned by policy evaluation, e.g., variants of TD($\lambda$) [163], using data collected from a dynamics model and the environment. Therefore, these methods can also be viewed as unbiased actor-critic approaches. In practice, it has been observed that these CVs indeed accelerate policy optimization, especially in simulated robot control tasks [147, 148, 149, 151, 162].

### 6.3 Why We Need New Control Variates

Given the decades-long development of CVs for policy gradient reviewed above, one might wonder if there is a need for new CV techniques. If so, what is the additional gain we can potentially have? To answer this question, let us first analyze the variance of policy gradient component $g_t$ and how the CVs above reduce it. By the law of total variance[‡], $\mathbb{V}[g_t]$ can be decomposed into *three terms*

$$\mathbb{V}_{s_t}\mathbb{E}_{|s_t}\left[\rho_t c_{t:T}\right] + \mathbb{E}_{s_t}\mathbb{V}_{a_t|s_t}\left[\rho_t \mathbb{E}_{|s_t,a_t}\left[c_{t:T}\right]\right] + \mathbb{E}_{s_t,a_t}\mathbb{V}_{|s_t,a_t}\left[\rho_t c_{t:T}\right], \tag{6.6}$$

where the first term is due to the randomness of policy and dynamics before getting to $s_t$, the second term is due to policy randomness alone at step $t$, i.e., selecting $a_t$, and the third term is due to again both the policy and the dynamics randomness in the future trajectories, i.e., after $s_t$ and $a_t$. We can measure the size of these three terms by their trace and define

$$\begin{gathered}\mathbb{V}_{s_t} := \mathrm{Tr}\left(\mathbb{V}_{s_t}\mathbb{E}_{|s_t}\left[\rho_t c_{t:T}\right]\right), \quad \mathbb{V}_{a_t|s_t} := \mathrm{Tr}\left(\mathbb{E}_{s_t}\mathbb{V}_{a_t|s_t}\left[\rho_t \mathbb{E}_{|s_t,a_t}\left[c_{t:T}\right]\right]\right), \\ \mathbb{V}_{|s_t,a_t} := \mathrm{Tr}\left(\mathbb{E}_{s_t,a_t}\mathbb{V}_{|s_t,a_t}\left[\rho_t c_{t:T}\right]\right).\end{gathered} \tag{6.7}$$

[‡]The law of total variance: $\mathbb{V}[f(x,y)] = \mathbb{E}_x\mathbb{V}_{y|x}[f(x,y)] + \mathbb{V}_x\mathbb{E}_{y|x}[f(x,y)]$ [164].

Hence, $\mathrm{Tr}\left(\mathbb{V}[g_t]\right) = \mathbb{V}_{s_t} + \mathbb{V}_{a_t|s_t} + \mathbb{V}_{|s_t,a_t}$. The following theorem shows the size of each term when the policy is Gaussian, which is commonly the case for problems with continuous actions.

**Theorem 5.** *Suppose the policy $\pi$ is Gaussian such that $\pi_{s_t}(a_t) = \mathcal{N}(\mu_\theta(s_t), \sigma I)$, where $\mu_\theta$ is the mean function, and $\theta$ and $\sigma > 0$ are learnable parameters. Assume the cost function $c$ is bounded and the Q-function $Q_\pi(s, a)$ is analytic in $a$. Then for small enough $\sigma$, it holds*

$$\mathbb{V}_{s_t} = O(h^2), \quad \mathbb{V}_{a_t|s_t} = O\left(\frac{h^2}{\sigma^4}\right), \quad \textit{and} \quad \mathbb{V}_{|s_t,a_t} = O\left(\frac{h^2}{\sigma^4}\right).$$

Here we focus on the effects due to the problem horizon $T$ and the policy variance $\sigma$. Theorem 5 shows that, when the stochasticity in policy decreases (e.g., when it passes the initial exploration phase) the terms $\mathbb{V}_{a_t|s_t}$ and $\mathbb{V}_{|s_t,a_t}$ will dominate variance in policy gradients. An intuitive explanation to this effect is that, as the policy becomes more deterministic, it becomes harder to approximate the derivative through zero-order feedback (i.e., accumulated costs). In particular, one can expect that $\mathbb{V}_{|s_t,a_t}$ is likely to be larger than $\mathbb{V}_{a_t|s_t}$ when the variation of $c_{t:T}$ is larger than the variation of $Q_\pi(s_t, a_t) = \mathbb{E}_{|s_t,a_t}[c_{t:T}]$. After understanding the composition of $\mathbb{V}[g_t]$, let us analyze $\mathbb{V}[\hat{g}_t^{\mathrm{SA}}]$ to see why using Q-function estimates as CVs (in Section Section 6.2.4) can reduce the variance.[§] Akin to the derivation of (6.6), one can show that $\mathbb{V}[\hat{g}_t^{\mathrm{SA}}]$ can be written as

$$\mathbb{V}_{s_t}\mathbb{E}_{|s_t}[\rho_t c_{t:T}] + \mathbb{E}_{s_t}\mathbb{V}_{a_t|s_t}[\rho_t(\mathbb{E}_{|s_t,a_t}[c_{t:T}] - \hat{q}_t)] + \mathbb{E}_{s_t,a_t}\mathbb{V}_{|s_t,a_t}[\rho_t c_{t:T}]. \tag{6.8}$$

Comparing (6.6) and (6.8), we can see that the CVs in the literature have been focusing on reducing *the second term* $\mathbb{V}_{a_t|s_t}$. Apparently, from the decomposition (6.8), the optimal choice of the state-action CV $\hat{Q}$ is the Q-function of the current policy $Q_\pi$, because $Q_\pi(s_t, a_t) = \mathbb{E}_{|s_t,a_t}[c_{t:T}]$, which explains why $\hat{Q}$ can be constructed by policy evaluation. When $\hat{Q} = Q_\pi$, the effect of $\mathbb{V}_{a_t|s_t}$ can be completely removed. In practice, $\hat{Q}$ is never

[§]Discussion on $\mathbb{V}[\hat{g}_t^{\mathrm{S}}]$ is omitted in that $\hat{g}_t^{\mathrm{S}}$ is subsumed by $\hat{g}_t^{\mathrm{SA}}$.

perfect (let alone the state-dependent version); nonetheless, improvement in learning speed has been consistently reported.

However, Theorem 5 suggests that $\mathbb{V}_{|s_t,a_t}$ can be in the similar magnitude as $\mathbb{V}_{a_t|s_t}$, implying that even when we completely remove the second term $\mathbb{V}_{a_t|s_t}$, the variance of the gradient estimate can still be significant. Indeed, recently Tucker et al. [149] empirically analyzed the three variance components in (6.8) in LQG and simulated robot locomotion tasks. They found that the third term $\mathbb{V}_{|s_t,a_t}$ is often close to the second term $\mathbb{V}_{a_t|s_t}$, and both of them are several orders of magnitude larger than the first term $\mathbb{V}_{s_t}$. Our Theorem 5 supports their finding and implies that there is a potential for improvement by reducing $\mathbb{V}_{|s_t,a_t}$. We discuss exactly how to do this next.

### 6.4 Trajectory-wise Control Variates

We propose a new family of trajectory-wise CVs, called TrajCV, that improves upon existing state or state-action CV techniques by tackling *additionally* $\mathbb{V}_{|s_t,a_t}$, the variance due to randomness in trajectory after step $t$ (cf. Section 6.3). While this idea sounds intuitively pleasing, a technical challenge immediately arises. Recall in designing CVs, we need to know the expectation of the proposed CV function over the randomness that we wish to reduce (see (6.3)). In this case, suppose we propose a CV $\phi(s_{t..T}, a_{t..T})$, we would need to know its conditional expectation $\mathbb{E}_{|s_t,a_t}[\phi(s_{t..T}, a_{t..T})]$. This need makes reducing $\mathbb{V}_{|s_t,a_t}$ fundamentally different from reducing $\mathbb{V}_{a_t|s_t}$, the latter of which has been the main focus in the literature: Because the dynamics $\mathcal{P}$ is unknown, we do not have access to the distribution of trajectories after step $t$ and therefore cannot compute $\mathbb{E}_{|s_t,a_t}$; by contrast, reducing $\mathbb{V}_{a_t|s_t}$ only requires knowing the policy $\pi$.

At first glance this seems like an impossible quest. But we will show that by a clever divide-and-conquer trick, an unbiased CV can actually be devised to reduce the variance $\mathbb{V}_{|s_t,a_t}$. The main idea is to *1)* decompose $\mathbb{V}_{|s_t,a_t}$ through repeatedly invoking the law of total variance and then *2)* attack the terms that are *amenable* to reduction using CVs. As expected,

the future variance cannot be completely removed, because of the unknown dynamics. But we should be able to reduce the randomness due to known distributions, namely, the future uses of policy $\pi$.

### 6.4.1 A Divide-and-Conquer Strategy

Before giving the details, let us first elucidate our idea using a toy problem. Consider estimating $\mathbb{E}\left[f(x_{1..5})\right]$, the expectation of a function $f$ of 5 random variables. We can apply the law of total variance repeatedly, in the order indicated by the subscript, and decompose the variance into

$$\mathbb{V}\left[f(x_{1..5})\right] = \sum_{k=1}^{5} \mathbb{E}_{x_{1..k-1}} \mathbb{V}_{x_k|x_{1..k-1}} \mathbb{E}_{x_{k+1..n}|x_{1..k}}[f(x_{1..5})] \tag{6.9}$$

For example, suppose we wish to reduce $\mathbb{V}_{x_3|x_{1..2}}$ we simply need to consider a CV in the form $\phi\left(x_{1..3}\right)$, which does not depends on random variables with larger indices. With the difference estimator $f(x_{1..5}) - \phi\left(x_{1..3}\right) + \mathbb{E}_{x_3|x_{1..2}}[\phi\left(x_{1..3}\right)]$, the variance $\mathbb{V}_{x_3|x_{1..2}}$ changes into $\mathbb{E}_{x_{1..2}} \mathbb{V}_{x_3|x_{1..2}}[\mathbb{E}_{x_{4..5}|x_{1..3}}[f(x_{1..5})] - \phi(x_{1..3})]$. Apparently when $\phi$ is optimally chosen as $\phi^\star(x_{1..3}) = \mathbb{E}_{x_{4..5}|x_{1..3}}[f(x_{1..5})]$, this term vanishes.

**Fact 1** A key of designing CVs by the recursive decomposition above is that the inclusion of the extra term, e.g., $\phi\left(x_{1..3}\right) - \mathbb{E}_{x_3|x_{1..2}}\left[\phi(x_{1..3})\right]$, in the difference estimator only affects a single component $\mathbb{V}_{x_3|x_{1..2}}$ in the total variance, *without influencing the other terms*. This separation property hence allows for a divide-and-conquer strategy: we can design CVs for each term separately and then combine them; the reduction on each term will add up and reduce the total variance.

**Fact 2** There is still one missing piece before we can adopt the above idea to design CVs for estimating policy gradients: the ordering of random variables. In the example above, we need to know $\mathbb{E}_{x_3|x_{1..2}}\left[\phi(x_{1..3})\right]$ to compute the difference estimator. Namely, it implicitly assumes the knowledge about $p(x_3|x_{1..2})$, which may or may not be accessible.

Suppose $p(x_3|x_{1..2})$ is not available but $p(x_3|x_{4..5})$ is. We can consider instead invoking the law of total variance in a different order, e.g., $x_4 \to x_5 \to x_3 \to x_1 \to x_2$, and utilize the information $p(x_3|x_{4..5})$ to construct a difference estimator to reduce $\mathbb{V}_{x_3|x_{4..5}}$. Therefore, the design of CVs hinges also on the information available. Recall that we only know about the policy but not the dynamics.

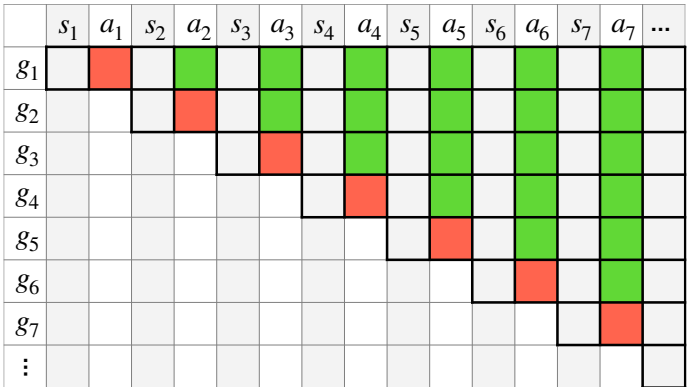



Figure 6.1: An illustration of the effects of state-action CV (6.5) and TrajCV (6.11). One row corresponds to one policy component $g_t = \rho_t c_{t:T}$, and thick borders indicate the random variables of which $g_t$ is a function. State-action CV reduces the variance due to the random variables in red, whereas TrajCV *additionally* affect the variance stemming from the random variables in green.

### 6.4.2 Design of TrajCV

After fleshing out the idea in the example above, we are now ready to present TrajCVs for policy gradient. Again we will focus on the component $g_t$ for transparency. Recall that $g_t$ is a function of $s_{t..T}$ and $a_{t..T}$. Given the information we know about these random variables (i.e., the policy) and the Markovian structure in MDP, a natural ordering of them for applying law of total variance is

$$s_t \to a_t \to s_{t+1} \to a_{t+1} \to \cdots \to s_T \to a_T. \tag{6.10}$$

Suppose now we want to reduce $\mathbb{V}_{a_k|s_{t..k},a_{t..k-1}}$ for some $k > t$. Based on Section 6.4.1, we may consider a CV in the form $\phi_k(s_{t..k}, a_{t..k})$, whose the optimal choice is $\phi_k^\star(s_{t..k}, a_{t..k}) =$

$\mathbb{E}_{|s_{t..k},a_{t..k}}[\rho_t c_{t:T}] = \rho_t\left(c_{t:k-1} + \mathbb{E}_{|s_{t..k},a_{t..k}}[c_{k:T}]\right) = \rho_t\left(c_{t:k-1} + Q_\pi(s_k,a_k)\right)$, where the last equality is due to the Markovian structure and the definition of $Q_\pi$. This suggests practically we can use $\phi_k(s_{t..k},a_{t..k}) = \rho_t(c_{t:k-1}+\hat{q}_k)$, where $\hat{q}_k := \hat{Q}(s_k,a_k)$ and $\hat{Q} \approx Q_\pi$ as was in (6.5).

In other words, we showed that finding the optimal CV for reducing variance in policy gradient can be reduced to learning a good Q-function estimate; this enables us to take advantage of existing policy evaluation algorithms. Now we combine[¶] $\{\phi_k(s_{t..k},a_{t..k})\}_{k=t}^T$ to build the CV for $g_t$. Because these terms do not interfere with each other (cf. Section 6.4.1), we can simply add them together into $\sum_{k=t}^T \phi_k(s_{t..k},a_{t..k})$ as the TrajCV. Equivalently, we have devised a difference estimator:

$$\begin{aligned}\hat{g}_t^{\mathrm{Traj}} &= g_t - \textstyle\sum_{k=t}^T \left(\phi_k(s_{t..k},a_{t..k}) - \mathbb{E}_{a_k|s_{t..k},a_{t..k-1}}[\phi_k(s_{t..k},a_{t..k})]\right)\\ &= g_t - \textstyle\sum_{k=t}^T \left(\rho_t\hat{q}_k - \mathbb{E}_{a_k|s_k}[\rho_t\hat{q}_k]\right)\end{aligned} \tag{6.11}$$

Comparing TrajCV in (6.11) and state-action CV in (6.4), we see that the state-action CV only contains the first term in the summation of TrajCV.[‖] The remaining terms with $k > t$ can be viewed as multiplying $\rho_t$ with estimates of future advantage functions: i.e., we have $\rho_t\hat{q}_k - \mathbb{E}_{a_k|s_k}[\rho_t\hat{q}_k] = \rho_t(\hat{q}_k - \mathbb{E}_{a_k|s_k}[\hat{q}_k])$. Appealing to law of total variance, $\mathbb{V}[\hat{g}_t^{\mathrm{Traj}}]$ can be decomposed into

$$\begin{aligned}&\underbrace{\mathbb{V}_{s_t}\mathbb{E}_{|s_t}[\rho_t c_{t:T}]}_{\text{due to } s_t} + \underbrace{\mathbb{E}_{s_t}\mathbb{V}_{a_t|s_t}[\rho_t(\mathbb{E}_{|s_t,a_t}[c_{t:T}] - \hat{q}_t)]}_{\text{due to } a_t} +\\ &\underbrace{\sum_{k=t}^T \mathbb{E}_{s_k,a_k}\mathbb{V}_{s_{k+1}|s_k,a_k}\left[\rho_t\mathbb{E}_{|s_{k+1}}[c_{k:T}]\right]}_{\text{due to dynamics randomness after step } t} + \underbrace{\sum_{k=t}^T \mathbb{E}_{s_{k+1}}\mathbb{V}_{a_{k+1}|s_{k+1}}[\rho_t(\mathbb{E}_{|s_{k+1},a_{k+1}}[c_{k:T}] - \hat{q}_{k+1})]}_{\text{due to policy randomness after step } t}\end{aligned} \tag{6.12}$$

where we further decompose the effect of $\mathbb{V}_{|s_t,a_t}$ in the second line into the randomness in dynamics and actions, respectively. Therefore, suppose the underlying dynamics is

[¶]When $k = t$, the CV is the same as state-action CV in (6.5), i.e., we define $\phi_k(s_t,a_t) = \rho_t\hat{q}_t$.

[‖]For brevity, we use CV to mean the difference estimator of that CV when there is no confusion.

deterministic (i.e., $\mathbb{V}_{s_{k+1}|s_k,a_k}$ vanishes), and $\hat{Q} = Q_\pi$ (e.g., when we have access to an accurate dynamics model), then using TrajCV (6.11) would *completely* remove $\mathbb{V}_{a_t|s_t}$ and $\mathbb{V}_{|s_t,a_t}$, the latter of which previous CVs (6.4) and (6.5) cannot affect. In Section 6.4.1, we visualize effects of TrajCV and state-action CV on each policy gradient component $g_t$. State-action CV only influences the diagonal terms, while TrajCV is able to affect the full upper-triangle parts. Note that in implementation of TrajCV for $g_{1:T}$, we only need to compute quantities $\hat{q}_t$, $\mathbb{E}_{a_t|s_t}[\hat{q}_t]$ and $\mathbb{E}_{a_t|s_t}[\rho_t \hat{q}_t]$ along a trajectory (done in $O(h)$ linear time) and they can be used to compute $\{\hat{g}_t^{\mathrm{Traj}}\}_{t=1}^T$ in (6.11). In addition, we remark that when $\hat{Q}(s,a) = \hat{V}(s)$, TrajCV reduces to the state-dependent CVs. Next we provide an example implementation of TrajCV.

### 6.4.3 Algorithm Example

Algorithm 5 specifies an instance of TrajCV, where Monte Carlo samples from a cheaper model simulator / dynamics model is used to approximate $\mathbb{E}_{a_t|s_t}[\hat{q}_t]$ (Line 9) and $\mathbb{E}_{a_t|s_t}[\rho_t \hat{q}_t]$ (Line 10).

In practice, the policy that's used for data collection may be different from the policy with respect to which the policy gradient is computed, e.g., when a whitening normalizer of the inputs to policy is updated after data collection or when off-policy samples are utilized. Here we derive a TrajCV that takes this into account. Let $\pi_{\mathrm{data}}$ be the data collection policy and $w_t := \frac{\pi_{s_t}(a_t)}{\pi_{\mathrm{data},s_t}(a_t)}$ be the importance weights, and define $w_{a\to b} := \prod_{k=a}^b w_k$ for $b \geq a$ and $w_{a\to b} = 1$ for $b < a$, akin to the symbol : that represents summation. Then we can write

$$\mathbb{E}_\pi[g_t] = \mathbb{E}_{\pi_{\mathrm{data}}}[g_t w_{t\to T}] = \mathbb{E}_{\rho_{\pi_{\mathrm{data}}}}\left[ g_t w_{t\to T} - \sum_{k=t}^{T} w_{t\to(k-1)} \left( w_k \rho_t \hat{q}_k - \mathbb{E}_{a_k\sim\pi_{s_k}}[\rho_t \hat{q}_k] \right) \right]$$

Note that this TrajCV is unbiased, and when $\hat{Q} = Q^\pi$, variance due to actions vanishes.

**Algorithm 5** Policy gradient estimate with TrajCV

**Input:** policy $\pi$, single trajectory by running $\pi$: $\{s_t, a_t, c_t\}_{t=1}^{T}$, value function estimate $\hat{V}$, deterministic dynamics estimate $\hat{d}$, number of action samples $N_a$
**Output:** policy gradient estimate $\hat{g}^{\mathrm{Traj}}$

**for** $t = 1$ **to** $T$ **do**
  $\hat{q}_t = \hat{V}(\hat{d}(s_t, a_t)) + c(s_t, a_t)$
  $\rho_t = \nabla \log \pi_{s_t}(a_t)$
  **for** $k = 1$ **to** $N_a$ **do**
    Sample $a'_k \sim \pi_{s_t}$
    $\hat{q}'_k = \hat{v}(\hat{d}(s_t, a'_k)) + c(s_t, a'_k)$
    $\rho'_k = \nabla \log \pi_{s_t}(a'_k)$
  **end for**
  $\hat{\mathbb{E}}[\hat{q}_t] = \frac{1}{N_a} \sum_{k=1}^{N_a} \hat{q}'_k$
  $\hat{\mathbb{E}}[\rho_t \hat{q}_t] = \frac{1}{N_a} \sum_{k=1}^{N_a} \rho'_k \hat{q}'_k$
**end for**

### 6.4.4 Optimality of the Natural Ordering

Recall in Section 6.4.1 we mentioned that the admissible ordering of random variables used in invoking the law of total variance depends on the information available. Here we show that the chosen ordering (6.10) is indeed the best ordering to adopt, as we only know the policy, not the dynamics.

We compare (6.10) against other potential orderings constructed by reparameterizing the policy such that its randomness in action becomes independent of the input state. We suppose the policy $\pi \in \Pi$ can be reparameterized by a function $\omega : \mathbb{S} \times \mathbb{R} \to \mathbb{A}$ and a distribution $p_\xi$, so that for all $s \in \mathbb{S}$, $\omega(s, \xi)$ and $\pi_s$ are equal. This is not a restricted assumption, which applies to, e.g., policies based on Gaussian [133, 100, 165] and Boltzmann [166, 134] distributions.

Reparameterization makes designing a larger family of TrajCVs possible. Because the component $g_t$ becomes a function of $\xi_{t..T}$ and $s_{t..T}$, the ordering the random variables in applying the law of total variance now can have many possibilities. In one extreme case, the

randomness of actions can be ordered before states (except $s_t$) as

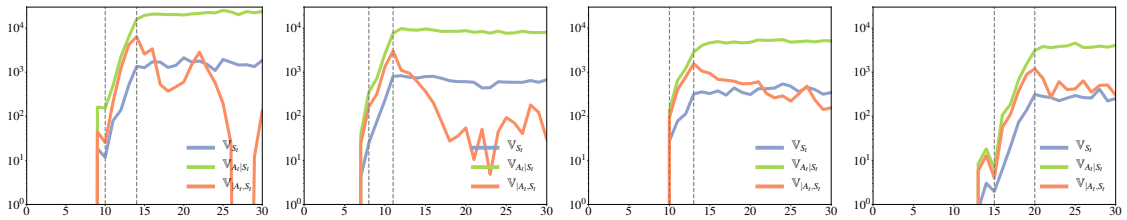
$$s_t \to \xi_t \to \cdots \to \xi_T \to s_{t+1} \to \cdots \to s_T, \tag{6.13}$$

leading to a CV that's a function of $\xi_{t..T}$ bearing the optimal choice $\mathbb{E}_{s_{t+1..h}|\xi_{t..T},s_t}\left[\rho_t c_{t:T}\right]$. Note that $\mathbb{E}_{s_{t+1..h}|\xi_{t..T},s_t}\left[\rho_t c_{t:T}\right]$ is a function that inputs the observable action randomness $\xi_{t..T}$, not the randomness of the unknown dynamics. Therefore, it can be approximated, e.g., if we have a biased simulator of the dynamics.** One might ask, given all possible orderings of random variables, which ordering we should pick to design the CV. Interestingly, to this question, the most natural one and the optimal one coincide. The proof is deferred to Appendix D.1.

**Theorem 6.** *Suppose that policy specified by $\omega$ and $p_\xi$ is known, but the dynamics $\mathcal{P}$ is unknown. Assume the optimal CV of a given ordering of random variables $s_{t..T}$ and $\xi_{t..T}$ can be obtained. Then the optimal ordering that minimizes the residue variance is the natural ordering in* (6.10) .

Theorem 6 tells us that if the optimal CVs are attainable (i.e., we can compute the Q-function exactly), then the natural ordering is optimal. However, we also remark that using this exact Q-function in the actor-critic rule can actually compute a gradient estimate that does not depend on any future randomness, which is better than using any of the CV techniques above as they yield noisy gradient estimates that always depend on unobservable randomness due to the stochastic dynamics. However, in practice, we can get neither the exact Q-function, nor the optimal CV of the other orderings. Consequently, further trade-off between bias and variance should be considered, which can have large effects in practice. For example, when using a biased Q-function estimate, it may be good to start with the the biased actor-critic gradient and then gradually switching to the unbiased TrajCV gradient as the step size decays. But considering the imperfection of the Q-function estimate, we may

**We sample all the action randomness $\xi_{t..T}$ first, execute the policy $\pi$ in simulation with fixed randomness $\xi_{t..T}$, and then collect the statistics $\rho_t c_{t:T}$.



(a) $\sigma = 0.01$ (b) $\sigma = 0.09$ (c) $\sigma = 1.0$ (d) $\sigma = 9.0$

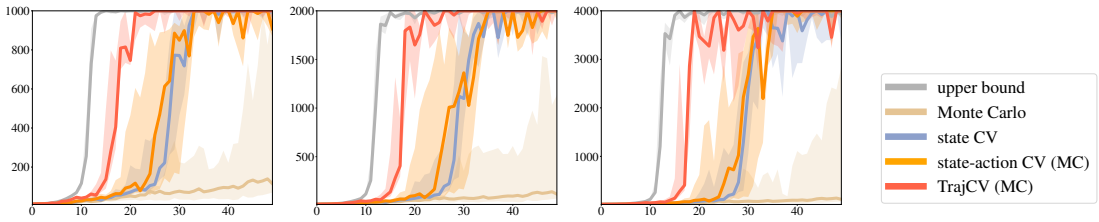
Figure 6.2: Components of $\mathrm{Tr}\left(\mathbb{V}[g_t]\right)$, for $t = 100$, evaluated at policies generated under the "upper bound" setting. $\sigma$ denotes the initial value of policy variance (defined in Theorem 5). The $x$-axis denotes the iteration number, and the $y$-axis is in $\log$ scale. The two vertical dashed lines mark the boundaries of iterations where the expected accumulated rewards is between $50$ and $900$.

also want to use other ordering instead of the natural one used in TrajCV. If the dynamics is relatively accurate and the computing resources for simulation are abundant, then although the residue is higher, the ordering (6.13) could actually be superior. The purpose of this chapter is to provide new insights into these different choices, but we leave further discussion on the bias-variance trade-off as an interesting practical question to pursue in future work. In the experiment section later, we will focus on the natural ordering (6.10).

## 6.5 Experimental Results and Discussion

Although the focus of this chapter is the theoretical insights, we illustrate our results with simulation of learning neural network policies to solve the CartPole balancing task in OpenAI Gym [97] powered by DART physics engine [98]. The policies are optimized using natural gradient descent [99]. Below we report in rewards, negative of costs, which is the natural performance measure provided in OpenAI Gym. The details of setup and implementation are provided in Appendix D.2.

First, in Figure 6.2, we corroborate the theoretical findings in Theorem 5 by empirically evaluating the components of $\mathrm{Tr}\left(\mathbb{V}[g_t]\right)$ during learning at $t = 100$. We observe that the learning process on CartPole can be partitioned into *three stages* (as delineated by the dashed vertical lines in Figure 6.2): 1) the initial exploration, where the policy performs




(a) $h = 1000$ (MC) (b) $h = 2000$ (MC) (c) $h = 4000$ (MC)

Figure 6.3: Results of naive Monte Carlo estimate, state CV, state-action CV, and TrajCV on CartPole problems with horizon $h = 1000, 2000, 4000$, where 5 trajectories are collected in each iteration. (MC) in the legend indicates that $\mathbf{E}_{a_t|s_t}$ in state-action CV and TrajCV is approximated by 1000 samples. "Upper bound" emulates the results of noiseless estimates with $100,000$ samples per iteration.[††] The $x$- and $y$-axes are iteration number and accumulated rewards, respectively. The median of 8 random seeds is plotted, and the shaded area accounts for 25% percentile.

very poorly and improves slowly, 2) the rapid improvement, where the policy performance increases steeply, and 3) the near convergence, where the policy reaches and stays at the peak performance. In the rapid improvement stage, due to the variance in the accumulated reward, $\mathbb{V}_{|s_t,a_t}$ is large, close to $\mathbb{V}_{a_t|s_t}$ and about 10 times of $\mathbb{V}_{s_t}$ as predicted by Theorem 5. Later on, when the policy is about to converge, $\mathbb{V}_{|s_t,a_t}$ drops because the performance becomes more consistent among trajectories especially for the case with small $\sigma$.

Next, in Figure 6.3, we compare naive Monte Carlo estimate (6.2), state-dependent CV (6.4), state-action CV (6.5), and TrajCV (6.11). We realized all algorithms with the same implementation of an on-policy value function approximator to facilitate a fair comparison; for the state-action CV and TrajCV, we used a Q-function estimate based on a biased physics simulator and the mentioned value function approximator. Overall, when more information is used to design the CVs (from state only, state-action, and then trajectory-wise) the convergence speed improves. In particular, as the problem horizon becomes longer, the gap becomes larger: the reward feedback becomes sparser, so the variance due to long-term trajectory starts to dominate, as shown in Figure 6.3c. In Appendix D.2, we provide further results of CVs based on other choices of Q-function approximators.

[††]For the usual learners, the number of samples collected per iteration is often less than $5T$, and much less at the start of learning, because of early termination when the agent fails.

These preliminary experimental results support the theoretical insights provided in Section 6.3 and Section 6.4, suggesting the importance of considering long-term effects in designing CVs, especially for problems with a long horizon. The fix turns out to be quite simple: just padding additional terms (cf. (6.11)) onto the existing CVs, which can be done using Q-function approximators available in existing CVs without new information. Interestingly we prove this simple idea is optimal. Important future work includes considering the different bias and variance trade-off discussed in Section 6.4.4.

# Part III

# Conclusion

# CHAPTER 7
# CONCLUSION AND DISCUSSION

Trajectory and policy optimization is a fundamental problem in robotics, and one clean mathematical formulation of this problem is stochastic optimization. As learning, solving stochastic optimization requires two ingredients: data and prior knowledge. The difficulties in gathering a large amount of data from physical interactions motivate us to design *general* methods that can exploit prior knowledge.

This thesis focused on one notable form of prior knowledge: dynamics models, which have been studied extensively in both control and machine learning communities. Utilizing the information in dynamics models is not straight-forward. Their computational demand restricts them to offline or batch use cases, and they can even impede the performance if the issue of modeling errors is not correctly addressed.

To this end, in Chapter 2 and Chapter 3 we developed general methods based on the idea of sparse Gaussian processes to achieve a desirable tradeoff between computation and performance for real-time incremental trajectory optimization. In Chapter 4, Chapter 5, Chapter 6, we developed two algorithmic and one theoretic foundation for designing unbiased accelerated policy optimization algorithms in the episodic setting.

### 7.0.1 Future Directions

Trajectory and policy optimization using dynamics models is a long-standing and ever-evolving challenge in robotics. In this section, we propose some potential directions to explore beyond the thesis.

**Bias versus variance in algorithms** In policy optimization, although it's desirable to design methods that are insensitive to modeling errors, being bias-free may not always

be the top priority. In the low data region, biased methods can outperform their unbiased counterparts. In some sense, there's a price we have to pay for being unbiased. This is in akin to the bias and variance tradeoff in statistical learning, but it is now under the policy learning context and pertains to the bias stemming from the algorithm in contrast to the hypothesis class. For instance, although control variates are unbiased, they require an accurate model to reduce variance. Therefore, when the model is imperfect and the gradient estimate is very noisy due to the limited number of samples, Studying this tradeoff more deeply to obtain the best of both worlds, probably under the framework of meta-learning, is an interesting future direction.

**Trajectory optimization versus policy optimization** In this thesis, we drew a distinct line between trajectory and policy optimization based on their different representations of trajectories (open-loop sequences versus reactive policies) and the different frequency of updates (per time step versus per several episodes). In fact, this line we've depicted can be quite fuzzy. In control, open-loop control sequence optimized in each step can be treated as a reactive policy that's parameterized by the dynamics model used to optimized the control sequence; policy can be updated in a receding horizon fashion: there's extensive research in reinforcement learning that investigates the single episode setting, e.g., eligibility traces. In spite of having the same formulation abstractly, in practice, trajectory optimization and and policy optimization approaches towards control have shown different weakness and strengths. In particular, trajectory optimization has been observed to be less sensitive to modelling error, probably due to online optimization. Exploring the causes that contribute to the practical differences and designing algorithms based on the new insights is a promising research direction.

**Ideas from multiple fields** Our research on (predictable) online policy optimization Chapter 4 and Chapter 5 benefits from previous research on online learning, reinforcement learning, statistical learning, and stochastic optimization. This suggests the opportunities of

bringing the novel ideas from different communities together to solving robotics problems in a theoretically-sound and principled way. Furthermore, this imposes great opportunities of new discoveries. For instance, in the predictable online learning project, we observed the tradeoff between computation and sample efficiency, i.e., spending more computation in each iteration improves sample efficiency. Since optimization mainly focuses on computation efficiency, and learning mainly focuses on sample efficiency, this observation suggests a new research direction that takes into account the two types of efficiency simultaneously.

**Formal study of the effect of model errors** Bridging the gap between simulation and reality has gained traction recently. However, designing heuristic solutions seems to be the main stream. I think we are lacking important theoretical underpinnings of the effect of model errors on policy performance. Although there are some preliminary results, e.g., that are dependent on the Lipschitz constants of dynamics, they are worst-case bounds, and can not provide meaningful insights on designing algorithms that can transfer policies learned in the simulation to the environment. Analyzing the optimization values of two different convex programs can be a good start.

**Pushing to more realistic settings** This thesis simplifies some of the problem settings to present the essential merits of the methods. For example, in general, we assume the state is fully observable and the cost functions are known; in online imitation learning, we assume the access to an algorithmic expert; for the experiments of policy optimization, they are mainly conducted in simulation. Although many of our insights are applicable to more general settings, some of them do not admit straight-forward extensions. For instance, the assumption of an algorithmic expert implies that the problem has almost already been solved. This assumption precludes the application of online imitation learning to a majority of imitation learning problems in the real world. Investigation of more realistic imitation learning settings, e.g.,multiple suboptimal experts and off-policy data, is a challenging and fruitful research direction.

# Appendices

# APPENDIX A

# APPENDIX FOR CHAPTER 3

## A.1 Prediction with uncertain input

In this section, we detail the two proposed methods of propagating uncertainty of $x$ through the probabilistic model $p(y|x)$, with the assumption that 1) the input $x$ is Gaussian distributed: $x \sim \mathcal{N}(\mu, \Sigma)$, and 2) the probabilistic model is represented as a SSGP. Through marginalization, the probability density function of the output is:

$$p(y) = \int p(y|x)p(x)\,\mathrm{d}x.$$

Computing this double integral exactly is intractable. Hence we apply analytically computed Gaussian approximation of the output distribution through: 1) exact moment matching, and 2) linearization.

### A.1.1 Preliminaries

We first provide some identities to facilitate later exposition of the closed-form expressions of exact moment matching (Section A.1.4) and linerization (Section A.1.5). We start with the derivatives for the feature map $\phi$ (3.4) using shorthand notation:

$$D\cos(\omega^\top x) = -\sin(\omega^\top x)\omega^\top = c_{d\omega}, \quad D\sin(\omega^\top x) = \cos(\omega^\top x)\omega^\top = s_{d\omega},$$

$$Dc_{d\omega} = -s_{d\omega}^\top \omega^\top, \quad Ds_{d\omega} = c_{d\omega}^\top \omega^\top,$$

$$D\phi(x) = M_x = \begin{bmatrix} D\phi^c(x) \\ D\phi^s(x) \end{bmatrix}, \quad D\phi_i^c(x) = \sigma_k c_{d\omega_i}, \quad D\phi_i^s(x) = \sigma_k s_{d\omega_i}.$$

Then Proposition 1 in the main text for the expectation of sinusoids over multivariate Gaussian distributions can be expressed as

$$\mathbb{E}[\cos(\omega^\top x)] = \exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\cos(\omega^\top \mu) = c_\omega$$
$$\mathbb{E}[\sin(\omega^\top x)] = \exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\sin(\omega^\top \mu) = s_\omega.$$

Proposition 2 for the expectation of the multiplication of sinusoids and linear functions over multivariate Gaussian distributions can be presented similarly:

$$\mathbb{E}\left[\cos(\omega^\top x)x\right] = c_\omega \mu - s_\omega \Sigma\omega = c_{x\omega}, \quad \mathbb{E}\left[\sin(\omega^\top x)x\right] = s_\omega \mu + c_\omega \Sigma\omega = s_{x\omega}.$$

We can further derive the expectations related to the feature map $\phi$ defined in (3.4):

$$\mathbb{E}[\phi(x)] = \psi, \quad \psi = \begin{bmatrix} \psi^c \\ \psi^s \end{bmatrix}, \quad \psi_i^c = \mathbb{E}[\phi_i^c(x)] = \sigma_k c_{\omega_i},$$
$$\mathbb{E}\left[\phi(x)\phi(x)^\top\right] = \Psi, \quad \Psi = \begin{bmatrix} \Psi^{cc} & \Psi^{cs} \\ \Psi^{sc} & \Psi^{ss} \end{bmatrix}, \quad \Psi_{ij}^{cc} = \frac{\sigma_k^2}{2}(c_{\omega_i+\omega_j} + c_{\omega_i-\omega_j}),$$
$$\Psi_{ij}^{cs} = \frac{\sigma_k^2}{2}(s_{\omega_i+\omega_j} - s_{\omega_i-\omega_j}), \quad \Psi_{ij}^{ss} = \frac{\sigma_k^2}{2}(-c_{\omega_i+\omega_j} + c_{\omega_i-\omega_j}),$$
$$\mathbb{E}\left[((\alpha^\top \phi(x))x)\right] = \Upsilon\alpha, \quad \Upsilon = \begin{bmatrix} \Upsilon_1^c & \dots & \Upsilon_m^c & \Upsilon_1^s & \dots & \Upsilon_m^s \end{bmatrix}, \quad \Upsilon_i^c = \sigma_k c_{x\omega_i}.$$

The derivatives of the preceding expectations to input statistics can be computed, with the notation $D_\mu f$ denoting the derivative of function $f$ with respect to $\mu$:

$$
\begin{aligned}
D_\mu c_\omega = -s_w \omega^\top, \quad D_\mu s_\omega = c_w \omega^\top, \\
D_\Sigma c_\omega = -\frac{1}{2} c_\omega \omega\omega^\top, \quad D_\Sigma s_\omega = -\frac{1}{2} s_\omega \omega\omega^\top, \\
D_\mu c_{x\omega} = \mu D_\mu c_\omega + c_\omega I - \Sigma\omega D_\mu s_\omega, \\
D_\mu s_{x\omega} = \mu D_\mu s_\omega + s_\omega I + \Sigma\omega D_\mu c_\omega, \\
D_\Sigma c_{x\omega} = \mu \otimes D_\Sigma c_\omega - \Sigma\omega \otimes D_\Sigma s_\omega - s_\omega (D_\Sigma \Sigma)\omega, \quad D_\Sigma \Sigma_{i,j} = J_{ji}, \\
D_\Sigma s_{x\omega} = \mu \otimes D_\Sigma s_\omega + \Sigma\omega \otimes D_\Sigma c_\omega + c_\omega (D_\Sigma \Sigma)\omega,
\end{aligned}
$$

where $J_{ij}$ is the matrix with all zeros except $ij$th entry being 1, $\otimes$ is tensor product, and assuming the operators' precedence: $D >$ matrix multiplication $> \otimes$.

### A.1.2 Proof for Proposition 1

The three useful identities involving quadratic exponentials to prove Proposition 1 and 2 are:

$$
\int \exp(-x^\top A x + v^\top x)\, \mathrm{d}x = \pi^{\frac{d}{2}} \det(A)^{-\frac{1}{2}} \exp(\frac{1}{4} v^\top A^{-1} v) = \eta, \tag{A.1}
$$

$$
\int (a^\top x) \exp(-x^\top A x + v^\top x)\, \mathrm{d}x = a^\top (\frac{1}{2} A^{-1} v)\eta, \tag{A.2}
$$

$$
p(x) = (2\pi)^{-\frac{d}{2}} \det(\Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2} \|x - \mu\|^2_{\Sigma^{-1}}) \quad x \sim \mathcal{N}(\mu, \Sigma). \tag{A.3}
$$

Proof for $\mathbb{E}[\cos(\omega^\top x)] = \exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\cos(\omega^\top \mu)$:

$$
\begin{aligned}
&\int \cos(\omega^\top x)\; p(x)\; \mathrm{d}x, \quad x \sim \mathcal{N}(\mu, \Sigma) \\
=&\mathrm{Re}\left(\int \left(\cos(\omega^\top x) + i\sin(\omega^\top x)\right)\; p(x)\; \mathrm{d}x\right) \\
=&\mathrm{Re}\left(\int \exp(i\omega^\top x)(2\pi)^{-\frac{d}{2}}\det(\Sigma)^{-\frac{1}{2}}\exp(-\frac{1}{2}(x-\mu)^\top\Sigma^{-1}(x-\mu))\mathrm{d}x\right) \qquad \text{(Use (A.3))} \\
=&\mathrm{Re}\left(\int \exp(i\omega^\top \mu)\exp(i\omega^\top(x-\mu))(2\pi)^{-\frac{d}{2}}\det(\Sigma)^{-\frac{1}{2}}\exp(-(x-\mu)^\top(2\Sigma)^{-1}(x-\mu))\mathrm{d}x\right) \\
=&(2\pi)^{-\frac{d}{2}}\det(\Sigma)^{-\frac{1}{2}}\mathrm{Re}\left(\exp(i\omega^\top\mu)\int \exp(-(x-\mu)^\top(2\Sigma)^{-1}(x-\mu) + (i\omega)^\top(x-\mu))\mathrm{d}x\right) \\
=&(2\pi)^{-\frac{d}{2}}\det(\Sigma)^{-\frac{1}{2}}\mathrm{Re}\left(\exp(i\omega^\top\mu)\pi^{\frac{d}{2}}\det(\frac{1}{2}\Sigma^{-1})^{-\frac{1}{2}}\exp(\frac{1}{4}(i\omega)^\top 2\Sigma(i\omega))\right) \qquad \text{(Use (A.1))} \\
=&\exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\cos(\omega^\top\mu).
\end{aligned}
$$

The other part of Proposition 1: $\mathbb{E}[\sin(\omega^\top x)] = \exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\sin(\omega^\top\mu)$ can be shown in a similar fashion, except that the imaginary operator $\mathrm{Im}$ will be used, instead of $\mathrm{Re}$.

### A.1.3 Proof for Proposition 2

To prove Proposition 2, we first prove that:

$$
\begin{aligned}
&\int a^\top x\;\cos(\omega^\top x)\; p(x)\; \mathrm{d}x, \quad x \sim \mathcal{N}(\mu, \Sigma) \\
=&\exp\left(-\frac{\omega^\top\Sigma\omega}{2}\right)\cos(\omega^\top\mu)a^\top\mu - \exp\left(-\frac{\omega^\top\Sigma\omega}{2}\right)\sin(\omega^\top\mu)a^\top\Sigma\omega.
\end{aligned}
\tag{A.4}
$$

$$
\begin{aligned}
&\int a^\top x \ \cos(\omega^\top x)\, p(x)\, \mathrm{d}x, \quad x \sim \mathcal{N}(\mu, \Sigma) \\
=&\mathrm{Re}\left(\int a^\top x \exp(i\omega^\top x)(2\pi)^{-\frac{n}{2}} \det(\Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2}\|x-\mu\|^2_{\Sigma^{-1}})\mathrm{d}x\right) \qquad \text{(Use (A.3))} \\
=&(2\pi)^{-\frac{n}{2}} \det(\Sigma)^{-\frac{1}{2}} \cdot \\
&\mathrm{Re}\left(\int \left(a^\top \mu + a^\top (x-\mu)\right) \exp(i\omega^\top \mu) \exp(i\omega^\top (x-\mu)) \exp(-\|x-\mu\|^2_{(2\Sigma)^{-1}})\mathrm{d}x\right) \\
=&a^\top \mu \exp(-\frac{1}{2}\|\omega\|^2_\Sigma) \cos(\omega^\top \mu) + \\
&(2\pi)^{-\frac{n}{2}} \det(\Sigma)^{-\frac{1}{2}} \mathrm{Re}\left(\left(\exp(i\omega^\top \mu) \int a^\top x \exp(-\frac{1}{2}\|x\|_{\Sigma^{-1}} + (i\omega)^\top x)\mathrm{d}x\right)\right. \\
=&a^\top \mu \exp(-\frac{1}{2}\|\omega\|^2_\Sigma) \cos(\omega^\top \mu) + \\
&(2\pi)^{-\frac{n}{2}} \det(\Sigma)^{-\frac{1}{2}} \mathrm{Re}\left(\exp(i\omega^\top \mu)\ i a^\top \Sigma\omega\ \pi^{\frac{n}{2}} \det(\frac{1}{2}\Sigma^{-1})^{-\frac{1}{2}} \exp(\frac{1}{4}\|i\omega\|^2_{2\Sigma})\right) \\
=&a^\top \mu \exp(-\frac{1}{2}\|\omega\|^2_\Sigma) \cos(\omega^\top \mu) - a^\top \Sigma\omega \exp(-\frac{1}{2}\omega^\top \Sigma\omega) \sin(\omega^\top \mu),
\end{aligned}
\tag{A.5}
$$

where the fourth equality uses (A.2). Then we can select $a$ as indicator vectors $e_i$ which contains zeros, except with the $i$th element being $1$. After stacking these elements together, we recover the identity of cosine part in Proposition 2. For the sine part, the same techniques apply.

### A.1.4 Exact moment matching

In exact moment matching, we match the mean and variance of the approximated Gaussian distribution with the ones of the true output distribution exactly. For simplicity of notations, henceforth, we suppress the dependency of $\phi(x)$ on $x$, and keep using $y$ for scalar function values. The predictive mean, variance, covariance between outputs, and cross-covariance

between inputs and outputs are

$$\mathbb{E}[y] = \mathbb{E}\mathbb{E}[y|x] = \mathbb{E}\left[\alpha^\top \phi\right] = \alpha^\top \psi,$$

$$\mathbb{V}[y] = \mathbb{E}\mathbb{V}[y|x] + \mathbb{V}\mathbb{E}[y|x] = \sigma_n^2 + \sigma_n^2 \mathbb{E}[\|\phi\|^2_{A^{-1}}] + \mathbb{E}[(\alpha^\top \phi)^2] - \mathbb{E}[y]^2$$

$$= \sigma_n^2 + \mathrm{Tr}\left(\underbrace{(\sigma_n^2 A^{-1} + \alpha\alpha^\top)}_{P} \Psi\right) - \mathbb{E}[y]^2,$$

$$\mathbb{V}[y_a, y_b] = \mathbb{V}\left[\mathbb{E}[y_a|x], \mathbb{E}[y_b|x]\right] = \alpha_a^\top \Psi_{ab} \alpha_b - \mathbb{E}[y_a]\mathbb{E}[y_b],$$

$$\mathbb{V}[x, y] = \mathbb{V}[x, \mathbb{E}[y|x]] = \mathbb{E}\left[x\mathbb{E}[y|x]\right] - \mathbb{E}[x]\mathbb{E}[y] = \Upsilon\alpha - \mathbb{E}[y]\mu,$$

where $\Psi_{ab}$ denotes that $\omega_i$ and $\omega_j$ in the definition of $\Psi$ come from the models for $y_a$ and $y_b$ respectively, since different output dimensions can have different feature maps, and the constant $\frac{\sigma_k^2}{2}$ is changed to $\frac{\sigma_{k,a}\sigma_{k,b}}{2}$ accordingly. The corresponding derivatives are derived using the chain rule:

$$D_\mu \mathbb{E}[y] = \alpha^\top D_\mu \psi,$$

$$D_\mu \mathbb{V}[y] = \mathrm{Tr}\left(P D_\mu \Psi\right) - 2\mathbb{E}[y] D_\mu \mathbb{E}[y],$$

$$D_\mu \mathbb{V}[y_a, y_b] = \alpha_a^\top (D_\mu \Psi_{ab}) \alpha_b - \mathbb{E}[y_a] D_\mu \mathbb{E}[y_b] - \mathbb{E}[y_b] D_\mu \mathbb{E}[y_a],$$

$$D_\mu \mathbb{V}[x, y] = (D_\mu \Upsilon)\alpha - \mathbb{E}[y] I - \mu D_\mu \mathbb{E}[y], \quad D_\Sigma \mathbb{V}[x, y] = (D_\Sigma \Upsilon)\alpha - \mu \otimes D_\Sigma \mathbb{E}[y],$$

where $I$ is an identity matrix with proper size. Substituting $D_\mu$ with $D_\Sigma$ yields the derivatives to the input covariance matrix $\Sigma$ if the expressions for $D_\Sigma$ are not explicitly provided above.

### A.1.5 Linearization

An alternative approach to Gaussian approximations of the predictive distribution is based on the linearization of the posterior mean function in (3.6) at the input mean $\mu$:

$$m(x) \approx m(\mu) + Dm(\mu)(x - \mu), \quad Dm(\mu) = \alpha^\top M_\mu.$$

Then the mean, (co)variance, and cross-covariance for the approximated Gaussian distribution can be computed as

$$\mathbb{E}[y] = \mathbb{E}\mathbb{E}[y|x] \approx \mathbb{E}\left[m(\mu) + \alpha^\top M(x-\mu)\right] = m(\mu),$$
$$\mathbb{V}[y] = \mathbb{E}\mathbb{V}[y|x] + \mathbb{V}\mathbb{E}[y|x] \approx \sigma_n^2 + \sigma_n^2\|\phi(\mu)\|^2_{A^{-1}} + \alpha^\top M\Sigma M^\top\alpha,$$
$$\mathbb{V}[y_a, y_b] = \mathbb{V}\left[\mathbb{E}[y_a|x], \mathbb{E}[y_b|x]\right] = \mathbb{E}\left[\alpha_a^\top M_a(x-\mu)(x-\mu)^\top M_b^\top\alpha_b\right] = \alpha_a^\top M_a\Sigma M_b^\top\alpha_b,$$
$$\mathbb{V}[x, y] = \mathbb{V}[x, \mathbb{E}[y|x]] = \mathbb{E}\left[(x-\mu)(\alpha^\top M(x-\mu))\right] = \alpha^\top M\Sigma,$$

where we omit the subscript $\mu$ for $M_\mu$, and $M_a$, $M_b$ stand for $M_\mu$ for model $y_a$, $y_b$ respectively. Their derivatives to the input statistics are

$$D_\mu\mathbb{E}[y] = \alpha^\top M, \quad D_\Sigma\mathbb{E}[y] = 0,$$
$$D_\mu\mathbb{V}[y] = 2\sigma_n^\top\phi(\mu)^\top A^{-1}M + 2\alpha^\top(D_\mu M)\Sigma M^\top\alpha, \quad D_\Sigma\mathbb{V}[y] = \alpha^\top M(D_\Sigma\Sigma)M^\top\alpha,$$
$$D_\mu\mathbb{V}[y_a, y_b] = \alpha_a^\top\left((D_\mu M_a)\Sigma M_b + M_a\Sigma(D_\mu M_b)\right)\alpha_b, \quad D_\Sigma\mathbb{V}[y_a, y_b] = \alpha_a^\top M_a(D_\Sigma\Sigma)M_b^\top\alpha_b,$$
$$D_\mu\mathbb{V}[x, y] = \alpha^\top(D_\mu M)\Sigma, \quad D_\Sigma\mathbb{V}[x, y] = \alpha^\top M(D_\Sigma\Sigma),$$

where for simplicity in notation, we omit the fact that approximation has been made in the equations for the derivatives.

## A.2 Discussions

### A.2.1 Conditional independence between outputs

To deal with multivariate outputs, the assumption of conditional independence between any two output dimensions is imposed, which implies that 1) the noise for different outputs are independent, e.g., Gaussian noise with diagonal covariance matrix, and 2) there's no cross-dependence between channels in the prior, e.g., a vector-valued Gaussian process (GP) prior with a matrix-valued kernel function that only has nonzero entries on the diagonal. These two conditions may be violated in practice. On one hand, the noise may not be

independent in general, e.g., wind blowing in some direction causes coupled noise on acceleration for aircraft. On the other hand, one may wish to exploit useful structure between different channels by incorporating them in the prior, e.g., dependence of velocity and acceleration in learning dynamics, and the relation between inverse dynamics models for different loads [167]. But, nevertheless, conditional independence is assumed in most of the related work [37, 38]. And we made this assumption in our experiments as well.

To accommodate conditional dependence in a principled way, vector-valued Gaussian processes can be used [53, 54]. This generally results in a more complicated model with additional computational cost. Incorporating vector-valued GPs would be an interesting extension of this work.

### A.2.2 SSGP-EKF vs. SSGP-ADF

In general, we hypothesize that SSGP-Lin is more sensitive to a small number of features than SSGP-EMM. In SSGP-Lin we use a locally linear approximation of a nonlinear function, and map a Gaussian distribution through this linear function. Intuitively, this locally linear approximation may vary significantly with the number of features, especially when the number of features is small. In contrast, in SSGP-EMM we compute the moments of the predictive distribution without this locally linear approximation. In order to validate this hypothesis, we performed additional experiments on a one-step approximate inference task, shown in Figure A.1. This exact phenomenon was observed in these experimental results. More precisely, when a small number of features are used (less than 20), the difference between SSGP-Lin and GP-Lin is greater than the difference between SSGP-EMM and GP-EMM, where the difference is measured by the KL divergence between the predictive distributions.

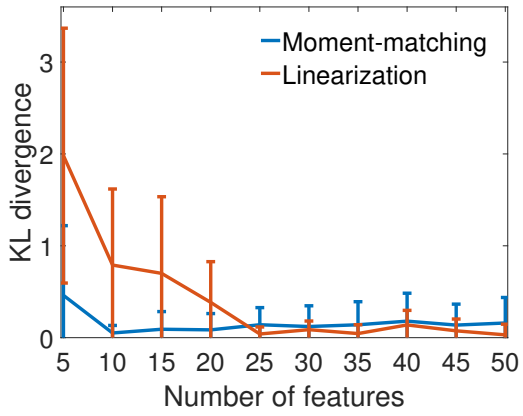



Figure A.1: KL divergences between SSGP-EMM and GP-EMM, SSGP-Lin and GP-Lin

# APPENDIX B

# APPENDIX FOR CHAPTER 4

## B.1 Relationship between PICCOLO and Existing Algorithms

We discuss how the framework of PICCOLO unifies existing online learning algorithms and provides their natural adaptive generalization. To make the presentation clear, we summarize the effective update rule of PICCOLO when the base algorithm is mirror descent

$$\begin{aligned} \pi_n &= \arg\min_{\pi \in \Pi} \langle w_n \hat{g}_n, \pi \rangle + B_{R_{n-1}}(\pi || \hat{\pi}_n) \\ \hat{\pi}_{n+1} &= \arg\min_{\pi \in \Pi} \langle w_n e_n, \pi \rangle + B_{R_n}(\pi || \pi_n) \end{aligned} \tag{B.1}$$

and that when the base algorithm is FTRL,

$$\begin{aligned} \pi_n &= \arg\min_{\pi \in \Pi} \langle w_n \hat{g}_n, \pi \rangle + \sum_{m=1}^{n-1} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi || \pi_m) \\ \hat{\pi}_{n+1} &= \arg\min_{\pi \in \Pi} \langle w_n e_n, \pi \rangle + B_{r_n}(\pi || \pi_n) + \langle w_n \hat{g}_n, \pi \rangle + \sum_{m=1}^{n-1} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi || \pi_m) \end{aligned} \tag{B.2}$$

Because $e_n = g_n - \hat{g}_n$, PICCOLO with FTRL exactly matches the update rule (MOBIL) [86]:

$$\begin{aligned} \pi_n &= \arg\min_{\pi \in \Pi} \langle w_n \hat{g}_n, \pi \rangle + \sum_{m=1}^{n-1} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi || \pi_m) \\ \hat{\pi}_{n+1} &= \arg\min_{\pi \in \Pi} \sum_{m=1}^{n} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi || \pi_m) \end{aligned} \tag{B.3}$$

As comparisons, we consider existing two-step update rules, which in our notation can be written as follows:

- Extragradient descent [168], mirror-prox [169, 91] or optimistic mirror descent [170,

81]

$$\begin{aligned}\pi_n &= \arg\min_{\pi\in\Pi} \langle \hat{g}_n, \pi\rangle + B_R(\pi||\hat{\pi}_n) \\ \hat{\pi}_{n+1} &= \arg\min_{\pi\in\Pi} \langle g_n, \pi\rangle + B_R(\pi||\hat{\pi}_n)\end{aligned} \tag{B.4}$$

- FTRL-with-Prediction/optimistic FTRL [81]

$$\pi_n = \arg\min_{\pi\in\Pi} R(\pi) + \langle \hat{g}_n, \pi\rangle + \sum_{m=1}^{n-1} \langle w_m g_m, \pi\rangle \tag{B.5}$$

Let us first review the previous update rules. Originally extragradient descent [168] and mirror prox [169, 91] were proposed to solve VIs (the latter is an extension to consider general Bregman divergences). As pointed out in [9], when applied to an online learning problem, these algorithms effectively assign $\hat{g}_n$ to be the online gradient as if the learner plays a decision at $\hat{\pi}_n$. On the other hand, in the online learning literature, optimistic mirror descent [170] was proposed to use $\hat{g}_n = g_{n-1}$. Later [81] generalized it to use some arbitrary sequence $\hat{g}_n$, and provided a FTRL version update rule in (B.5). However, it is unclear in [81] where the prediction $\hat{g}_n$ comes from in general, though they provide an example in the form of learning from experts.

Recently the FTRL version of these ideas is generalized to design MOBIL [86], which introduces extra features 1) use of weights 2) non-stationary Bregman divergences (i.e.,step size) and 3) the concept of predictive models ($\Phi_n \approx \nabla l_n$). The former two features are important to speed up the convergence rate of IL. With predictive models, they propose a conceptual idea (inspired by Be-the-Leader) which solves for $\pi_n$ by the VI of finding $\pi_n$ such that

$$\left\langle w_n \Phi_n(\pi_n) + \sum_{m=1}^{n} w_m g_m, \pi' - \pi_n \right\rangle \geq 0 \qquad \forall \pi' \in \Pi \tag{B.6}$$

and a more practical version (B.3) which sets $\hat{g}_n = \Phi_n(\pi_n)$. Under proper assumptions, they

prove that the practical version achieves the same rate of non-asymptotic convergence as the conceptual one, up to constant factors.

PICCOLO unifies and generalizes the above update rules. We first notice that when the weight is constant, the set $\Pi$ is unconstrained, and the Bregman divergence is constant, PICCOLO with mirror descent in (B.1) is the same as (B.4), i.e.,

$$\begin{aligned}
\hat{\pi}_{n+1} &= \arg\min_{\pi\in\Pi} \langle e_n, \pi\rangle + B_R(\pi||\pi_n) \\
&= \arg\min_{\pi\in\Pi} \langle e_n, \pi\rangle + R(\pi) - \langle \nabla R(\pi_n), \pi\rangle \\
&= \arg\min_{\pi\in\Pi} \langle g_n - \hat{g}_n, \pi\rangle + R(\pi) - \langle \nabla R(\hat{\pi}_n) - \hat{g}_n, \pi\rangle \\
&= \arg\min_{\pi\in\Pi} \langle g_n, \pi\rangle + R(\pi) - \langle \nabla R(\hat{\pi}_n), \pi\rangle \\
&= \arg\min_{\pi\in\Pi} \langle g_n, \pi\rangle + B_R(\pi||\hat{\pi}_n)
\end{aligned}$$

Therefore, PICCOLO with mirror descent includes previous two-step algorithms with proper choices of $\hat{g}_n$. On the other hand, we showed above that PICCOLO with FTRL (B.2) recovers exactly (B.3).

PICCOLO further generalizes these updates in two important aspects. First, it provides a systematic way to make these mirror descent and FTRL algorithms *adaptive*, by the reduction that allows reusing existing adaptive algorithm designed for adversarial settings. By contrast, all the previous update schemes discussed above (even MOBIL) are based on constant or pre-scheduled Bregman divergences, which requires the knowledge of several constants of problem properties that are usually unknown in practice. The use of adaptive schemes more amenable to hyperparameter tuning in practice.

Second, PICCOLO generalize the use of predictive models from the VI formulation in (B.6) to the *fixed-point* formulation in (4.8). One can show that when the base algorithm is FTRL and we remove the Bregman divergence*, (4.8) is the same as (B.6). In other words,

*Originally the conceptual MOBIL algorithm is based on the assumption that $l_n$ is strongly convex and therefore does not require extra Bregman divergence. Here PICCOLO with FTRL provides a natural

(B.6) essentially can be viewed as a mechanism to find $\hat{g}_n$ for (B.3). But importantly, the fixed-point formulation is method agnostic and therefore applies to also the mirror descent case. In particular, in Section 4.4.2, we point out that when $\Phi_n$ is a gradient map, the fixed-point problem reduces to finding a stationary point[†] of a non-convex optimization problem. This observation makes implementation of the fixed-point idea much easier and more stable in practice (as we only require the function associated with $\Phi_n$ to be lower bounded to yield a stable problem).

## B.2 Proof of Lemma 3

Without loss of generality we suppose $w_1 = 1$ and $J(\pi) \geq 0$ for all $\pi$. And we assume the weighting sequence $\{w_n\}$ satisfies, for all $n \geq m \geq 1$ and $k \geq 0$, $\frac{w_{n+k}}{w_n} \leq \frac{w_{m+k}}{w_m}$. This means $\{w_n\}$ is an non-decreasing sequence and it does not grow faster than exponential (for which $\frac{w_{n+k}}{w_n} = \frac{w_{m+k}}{w_m}$). For example, if $w_n = n^p$ with $p \geq 0$, it easy to see that

$$\frac{(n+k)^p}{n^p} \leq \frac{(m+k)^p}{m^p} \impliedby \frac{n+k}{n} \leq \frac{m+k}{m} \impliedby \frac{k}{n} \leq \frac{k}{m}$$

For simplicity, let us first consider the case where $l_n$ is deterministic. Given this assumption, we bound the performance in terms of the weighted regret below. For $l_n$ defined in (4.5), we

generalization to online convex problems.

[†]Any stationary point will suffice.

can write

$$
\begin{aligned}
&\sum_{n=1}^{N} w_n J(\pi_n) \\
&= \sum_{n=1}^{N} w_n J(\pi_{n-1}) + w_n \mathbb{E}_{d_{\pi_n}} \mathbb{E}_{\pi_n}[A_{\pi_{n-1}}] \\
&= \sum_{n=1}^{N} w_n J(\pi_{n-1}) + w_n l_n(\pi_n) \\
&= w_1 J(\pi_0) + \sum_{n=1}^{N-1} w_{n+1} J(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \\
&= w_1 J(\pi_0) + \sum_{n=1}^{N-1} w_{n+1} J(\pi_{n-1}) + \sum_{n=1}^{N-1} w_{n+1} l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \\
&= (w_1 + w_2) J(\pi_0) + \sum_{n=1}^{N-2} w_{n+2} J(\pi_n) + \sum_{n=1}^{N-1} w_{n+1} l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \\
&= w_{1:N} J(\pi_0) + \left( w_N l_1(\pi_1) + \sum_{n=1}^{2} w_{n+N-2} l_n(\pi_n) + \cdots + \sum_{n=1}^{N-1} w_{n+1} l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \right) \\
&= w_{1:N} J(\pi_0) + \\
&\left( w_N l_1(\pi_1) + \sum_{n=1}^{2} \frac{w_{n+N-2}}{w_n} w_n l_n(\pi_n) + \cdots + \sum_{n=1}^{N-1} \frac{w_{n+1}}{w_n} w_n l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \right) \\
&\leq w_{1:N} J(\pi_0) + \\
&\left( w_N l_1(\pi_1) + \frac{w_{N-1}}{w_1} \sum_{n=1}^{2} w_n l_n(\pi_n) + \cdots + \frac{w_2}{w_1} \sum_{n=1}^{N-1} w_n l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \right) \\
&= w_{1:N} J(\pi_0) + \\
&\left( w_N l_1(\pi_1) + w_{N-1} \sum_{n=1}^{2} w_n l_n(\pi_n) + \cdots + w_2 \sum_{n=1}^{N-1} w_n l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \right)
\end{aligned}
$$

where the inequality is due to the assumption on the weighting sequence.

We can further rearrange the second term in the final expression as

$$
\begin{aligned}
& w_N l_1(\pi_1) + w_{N-1} \sum_{n=1}^{2} w_n l_n(\pi_n) + \cdots + w_2 \sum_{n=1}^{N-1} w_n l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \\
=& w_N \left( l_1(\pi_1) - \min_{\pi \in \Pi} l_1(\pi) + \min_{\pi \in \Pi} l_1(\pi) \right) \\
& + w_{N-1} \left( \sum_{n=1}^{2} w_n l_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{2} w_n l_n(\pi) + \min_{\pi \in \Pi} \sum_{n=1}^{2} w_n l_n(\pi) \right) \\
& + \cdots + \sum_{n=1}^{N} w_n l_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{N} w_n l_n(\pi) + \min_{\pi \in \Pi} \sum_{n=1}^{N} w_n l_n(\pi) \\
=& \sum_{n=1}^{N} w_{N-n+1} \left( \mathrm{Regret}_n(f) + w_{1:n} \epsilon_n(f) \right)
\end{aligned}
$$

where the last equality is due to the definition of *static* regret and $\epsilon_n$.

Likewise, we can also write the above expression in terms of *dynamic regret*

$$
\begin{aligned}
& w_N l_1(\pi_1) + w_{N-1} \sum_{n=1}^{2} w_n l_n(\pi_n) + \cdots + w_2 \sum_{n=1}^{N-1} w_n l_n(\pi_n) + \sum_{n=1}^{N} w_n l_n(\pi_n) \\
=& w_N \left( l_1(\pi_1) - \min_{\pi \in \Pi} l_1(\pi) + \min_{\pi \in \Pi} l_1(\pi) \right) \\
& + w_{N-1} \left( \sum_{n=1}^{2} w_n l_n(\pi_n) - \sum_{n=1}^{2} w_n \min_{\pi \in \Pi} l_n(\pi) + \sum_{n=1}^{2} \min_{\pi \in \Pi} w_n l_n(\pi) \right) \\
& + \cdots + \sum_{n=1}^{N} w_n l_n(\pi_n) - \sum_{n=1}^{N} \min_{\pi \in \Pi} w_n l_n(\pi) + \sum_{n=1}^{N} \min_{\pi \in \Pi} w_n l_n(\pi) \\
=& \sum_{n=1}^{N} w_{N-n+1} \left( \mathrm{Regret}_n^d(l) + w_{1:n} \epsilon_n^d(l) \right)
\end{aligned}
$$

in which we define the weighted dynamic regret as

$$
\mathrm{Regret}_n^d(l) = \sum_{m=1}^{n} w_m l_m(\pi_m) - \sum_{m=1}^{n} w_m \min_{\pi \in \Pi} l_m(\pi)
$$

and an expressive measure based on dynamic regret

$$\epsilon_n^d = \frac{1}{w_{1:n}} \sum_{m=1}^{n} w_m \min_{\pi \in \Pi} l_m(\pi) \leq 0$$

For stochastic problems, because $\pi_n$ does not depends on $\tilde{l}_n$, the above bound applies to the performance in expectation. Specifically, let $h_{n-1}$ denote all the random variables observed before making decision $\pi_n$ and seeing $\tilde{l}_n$. As $\pi_n$ is made independent of $\tilde{l}_n$, we have, for example,

$$\begin{aligned}
\mathbb{E}[l_n(\pi_n)|h_{n-1}] &= \mathbb{E}[l_n(\pi_n)|h_{n-1}] - \mathbb{E}[l_n(\pi_n^*)|h_{n-1}] + \mathbb{E}[l_n(\pi_n^*)|h_{n-1}] \\
&= \mathbb{E}[\tilde{l}_n(\pi_n)|h_{n-1}] - \mathbb{E}[\tilde{l}_n(\pi_n^*)|h_{n-1}] + \mathbb{E}[l_n(\pi_n^*)|h_{n-1}] \\
&\leq \mathbb{E}[\tilde{l}_n(\pi_n) - \min_{\pi \in \Pi} \tilde{l}_n(\pi)|h_{n-1}] + \mathbb{E}[l_n(\pi_n^*)|h_{n-1}]
\end{aligned}$$

where $\pi_n^* = \arg\min_{\pi \in \Pi} l_n(\pi)$. By applying a similar derivation as above recursively, we can extend the previous deterministic bounds to bounds in expectation (for both the static or the dynamic regret case), proving the desired statement.

### B.3 The Basic Operations of Base Algorithms

We provide details of the abstract basic operations shared by different base algorithms. In general, the update rule of any base mirror-descent or FTRL algorithm can be represented in terms of the three basic operations

$$h \leftarrow \texttt{update}(h, H, g, w), \qquad H \leftarrow \texttt{adapt}(h, H, g, w), \qquad \pi \leftarrow \texttt{project}(h, H) \quad \text{(B.7)}$$

where `update` and `project` can be identified standardly, for mirror descent as,

$$\texttt{update}(h, H, g, w) = \arg\min_{\pi' \in \Pi} \langle wg, \pi' \rangle + B_H(\pi||h), \qquad \texttt{project}(h, H) = h \quad \text{(B.8)}$$

and for FTRL as,

$$\texttt{update}(h, H, g, w) = h + wg, \qquad \texttt{project}(h, H) = \arg\min_{\pi' \in \Pi} \langle h, \pi' \rangle + H(\pi') \tag{B.9}$$

We note that in the main text the operation `project` is omitted for simplicity, as it is equal to the identify map for mirror descent. In general, it represents the decoding from the abstract representation of the decision $h$ to $\pi$. The main difference between and $h$ and $\pi$ is that $h$ represents the sufficient information that defines the state of the base algorithm.

While `update` and `project` are defined standardly, the exact definition of `adapt` depends on the specific base algorithm. Particularly, `adapt` may depend also on whether the problem is weighted, as different base algorithms may handle weighted problems differently. Based on the way weighted problems are handled, we roughly categorize the algorithms (in both mirror descent and FTRL families) into two classes: the *stationary* regularization class and the *non-stationary* regularization class. Here we provide more details into the algorithm-dependent `adapt` operation, through some commonly used base algorithms as examples.

Please see also Appendix B.1 for connection between PICCOLO and existing two-step algorithms, like optimistic mirror descent [171].

### B.3.1 Stationary Regularization Class

The `adapt` operation of these base algorithms features two major functions: 1) a moving-average adaptation and 2) a step-size adaption. The moving-average adaptation is designed to estimate some statistics $G$ such that $\|g\|_* = O(G)$ (which is an important factor in regret bounds), whereas the step-size adaptation updates a scalar multiplier $\eta$ according to the weight $w$ to ensure convergence.

This family of algorithms includes basic mirror descent [154] and FTRL [172, 96] with a scheduled step size, and adaptive algorithms based on moving average e.g., RMSPROP [173]

ADADELTA [174], ADAM [95], AMSGRAD [175], and the adaptive NATGRAD we used in the experiments. Below we showcase how `adapt` is defined using some examples.

*Basic mirror descent [154]*

We define $G$ to be some constant such that $G \geq \sup \|g_n\|_*$ and define

$$\eta_n = \frac{\eta}{1 + cw_{1:n}/\sqrt{n}}, \tag{B.10}$$

as a function of the iteration counter $n$, where $\eta > 0$ is a step size multiplier and $c > 0$ determines the decaying rate of the step size. The choice of hyperparameters $\eta$, $c$ pertains to how far the optimal solution is from the initial condition, which is related to the size of $\Pi$. In implementation, `adapt` updates the iteration counter $n$ and updates the multiplier $\eta_n$ using $w_n$ in (B.10).

Together $(n, G, \eta_n)$ defines $H_n = R_n$ in the mirror descent update rule (4.6) through setting $R_n = \frac{G}{\eta_n} R$, where $R$ is a strongly convex function. That is, we can write (4.6) equivalently as

$$\begin{aligned}
\pi_{n+1} &= \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + \frac{G}{\eta_n} B_R(\pi || \pi_n) \\
&= \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + B_{H_n}(\pi || \pi_n) \\
&= \texttt{update}(h_n, H_n, g_n, w_n)
\end{aligned}$$

When the weight is constant (i.e.,$w_n = 1$), we can easily see this update rule is equivalent to the classical mirror descent with a step size $\frac{\eta/G}{1+c\sqrt{n}}$, which is the optimal step size [96]. For general $w_n = \Theta(n^p)$ with some $p > -1$, it can viewed as having an effective step size $\frac{w_n \eta_n}{G} = O(\frac{1}{G\sqrt{n}})$, which is optimal in the weighted setting. The inclusion of the constant $G$ makes the algorithm invariant to the scaling of loss functions. But as the same $G$ is used across all the iterations, the basic mirror descent is conservative.

*Basic FTRL [96]*

We provide details of general FTRL

$$\pi_{n+1} = \arg\min_{\pi \in \Pi} \sum_{m=1}^{n} \langle g_m, \pi \rangle + B_{r_m}(\pi || \pi_m) \tag{B.11}$$

where $B_{r_m}(\cdot||\pi_m)$ is a Bregman divergence centered at $\pi_m$.

We define, in the $n$th iteration, $h_n$, $H_n$, and `project` of FTRL in (B.9) as

$$h_n = \sum_{m=1}^{n} w_m g_m, \qquad H_n(\pi) = \sum_{m=1}^{n} B_{r_m}(\pi||\pi_n),$$
$$\texttt{project}(h, H) = \arg\min_{\pi' \in \Pi} \langle h, \pi' \rangle + H(\pi')$$

Therefore, we can see that $\pi_{n+1} = \texttt{project}(h_n, H_n)$ indeed gives the update (B.11):

$$\begin{aligned}
\pi_{n+1} &= \texttt{project}(h_n, H_n) \\
&= \texttt{project}(\sum_{m=1}^{n} w_m g_m, \sum_{m=1}^{n} B_{r_m}(\pi||\pi_n)) \\
&= \arg\min_{\pi \in \Pi} \sum_{m=1}^{n} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi||\pi_m)
\end{aligned}$$

For the basic FTRL, the `adapt` operator is similar to the basic mirror descent, which uses a constant $G$ and updates the memory $(n, \eta_n)$ using (B.10). The main differences are how $(G, \eta_n)$ is mapped to $H_n$ and that the basic FTRL updates $H_n$ also using $h_n$ (i.e.,$\pi_n$). Specifically, it performs $H_n \leftarrow \texttt{adapt}(h_n, H_{n-1}, g_n, w_n)$ through the following:

$$H_n(\cdot) = H_{n-1}(\cdot) + B_{r_n}(\cdot||\pi_n)$$

where following [96] we set

$$B_{r_n}(\pi||\pi_n) = G(\frac{1}{\eta_n} - \frac{1}{\eta_{n-1}})B_R(\pi||\pi_n)$$

and $\eta_n$ is updated using some scheduled rule.

One can also show that the choice of $\eta_n$ scheduling in (B.10) leads to an optimal regret. When the problem is uniformly weighted (i.e.,$w_n = 1$), this gives exactly the update rule in [96]. For general $w_n = \Theta(n^p)$ with $p > -1$, a proof of optimality can be found, for example, in the appendix of [9].

### ADAM *[95] and* AMSGRAD *[175]*

As a representing mirror descent algorithm that uses moving-average estimates, ADAM keeps in the memory of the statistics of the first-order information that is provided in `update` and `adapt`. Here we first review the standard description of ADAM and then show how it is summarized in

$$H_n = \texttt{adapt}(h_n, H_{n-1}, g_n, w_n), \qquad h_{n+1} = \texttt{update}(h_n, H_n, g_n, w_n) \tag{4.7}$$

using properly constructed `update`, `adapt`, and `project` operations.

The update rule of ADAM [95] is originally written as, for $n \geq 1$,[‡]

$$\begin{aligned} m_n &= \beta_1 m_{n-1} + (1-\beta_1) g_n \\ v_n &= \beta_2 v_{n-1} + (1-\beta_2) g_n \odot g_n \\ \hat{m}_n &= m_n/(1-\beta_1^n) \\ \hat{v}_n &= v_n/(1-\beta_2^n) \\ \pi_{n+1} &= \pi_n - \eta_n \hat{m}_n \oslash (\sqrt{\hat{v}_n} + \epsilon) \end{aligned} \tag{B.12}$$

[‡]We shift the iteration index so it conforms with our notation in online learning, in which $\pi_1$ is the initial policy before any update.

where $\eta_n > 0$ is the step size, $\beta_1, \beta_2 \in [0, 1)$ (default $\beta_1 = 0.9$ and $\beta_2 = 0.999$) are the mixing rate, and $0 < \epsilon \ll 1$ is some constant for stability (default $\epsilon = 10^{-8}$), and $m_0 = v_0 = 0$. The symbols $\odot$ and $\oslash$ denote element-wise multiplication and division, respectively. The third and the forth steps are designed to remove the 0-bias due to running moving averages starting from 0.

The above update rule can be written in terms of the three basic operations. First, we define the memories $h_n = (m_n, \pi_n)$ for policy and $(v_n, \eta_n, n)$ for regularization that is defined as

$$H_n(\pi) = \frac{1}{2\eta_n} \pi^\top (\mathrm{diag}(\sqrt{\hat{v}_n}) + \epsilon I)\pi \tag{B.13}$$

where $\hat{v}_n$ is defined in the original ADAM equation in (B.12).

The `adapt` operation updates the memory to $(v_n, \eta_n, n)$ in the step

$$H_n \leftarrow \texttt{adapt}(h_n, H_{n-1}, g_n, w_n)$$

It updates the iteration counter $n$ and $\eta_n$ in the same way in the basic mirror descent using (B.10), and update $v_n$ (which along with $n$ defines $\hat{v}_n$ used in (B.13)) using the original ADAM equation in (B.12).

For `update`, we slightly modify the definition of `update` in (B.8) (replacing $g_n$ with $\hat{m}_n$) to incorporate the moving average and write

$$\texttt{update}(h_n, H_n, g_n, w_n) = \arg\min_{\pi' \in \Pi} \langle w_n \hat{m}_n, \pi' \rangle + B_{H_n}(\pi' || \pi) \tag{B.14}$$

where $m_n$ and $\hat{m}_n$ are defined the same as in the original ADAM equations in (B.12). One can verify that, with these definitions, the update rule in (4.7) is equivalent to the update rule (B.12), when the weight is uniform (i.e.,$w_n = 1$).

Here the $\sqrt{\hat{v}_n}$ plays the role of $G$ as in the basic mirror descent, which can be viewed as

an estimate of the upper bound of $\|g_n\|_*$. ADAM achieves a better performance because a coordinate-wise online estimate is used. With this equivalence in mind, we can easily deduct that using the same scheduling of $\eta_n$ as in the basic mirror descent would achieve an optimal regret (cf. [95, 175]). We note that ADAM may fail to converge in some particular problems due to the moving average [175]. AMSGRAD [175] modifies the moving average and uses strictly increasing estimates. However in practice AMSGRAD behaves more conservatively.

For weighted problems, we note one important nuance in our definition above: it separates the weight $w_n$ from the moving average and considers $w_n$ as part of the $\eta_n$ update, because the growth of $w_n$ in general can be much faster than the rate the moving average converges. In other words, the moving average can only be used to estimate a stationary property, not a time-varying one like $w_n$. Hence, we call this class of algorithms, the *stationary* regularization class.

*Adaptive* NATGRAD

Given first-order information $g_n$ and weight $w_n$, we consider an update rule based on Fisher information matrix:

$$\pi_{n+1} = \arg\min_{\pi\in\Pi} \langle w_n g_n, \pi\rangle + \frac{\sqrt{\hat{G}_n}}{2\eta_n}(\pi-\pi_n)^\top F_n(\pi-\pi_n) \tag{B.15}$$

where $F_n$ is the Fisher information matrix of policy $\pi_n$ [176] and $\hat{G}_n$ is an adaptive multiplier for the step size which we will describe. When $\hat{G}_n = 1$, the update in (B.15) gives the standard natural gradient descent update with step size $\eta_n$ [99] .

The role of $\hat{G}_n$ is to adaptively and *slowly* changes the step size to minimize $\sum_{n=1}^N \frac{\eta_n}{\sqrt{G_n}}\|g_n\|^2_{F_n,*}$, which plays an important part in the regret bound (see Section 4.5, Appendix B.6, and

e.g. [96] for details). Following the idea in ADAM, we update $\hat{G}_n$ by setting (with $G_0 = 0$)

$$\begin{aligned} G_n &= \beta_2 G_n + (1-\beta_2)\frac{1}{2} g_n^\top F_n^{-1} g_n \\ \hat{G}_n &= G_n/(1-\beta_2^n) \end{aligned} \tag{B.16}$$

similar to the concept of updating $v_n$ and $\hat{v}_n$ in ADAM in (B.12), and update $\eta_n$ in the same way as in the basic mirror descent using (B.10). Consequently, this would also lead to a regret like ADAM but in terms of a different local norm.

The `update` operation of adaptive NATGRAD is defined standardly in (4.6) (as used in the experiments). The `adapt` operation updates $n$ and $\eta_n$ like in ADAM and updates $G_n$ through (B.16).

### B.3.2 Non-Stationary Regularization Class

The algorithms in the non-stationary regularization class maintains a regularization that is increasing over the number of iterations. Notable examples of this class include ADAGRAD [82] and ONLINE NEWTON STEP [177], and its regularization function is updated by applying BTL in a secondary online learning problem whose loss is an upper bound of the original regret (see [178] for details). Therefore, compared with the previous stationary regularization class, the adaption property of $\eta_n$ and $G_n$ exchanges: $\eta_n$ here becomes constant and $G_n$ becomes time-varying. This will be shown more clearly in the ADAGRAD example below. We note while these algorithms are designed to be optimal in the convex, they are often too conservative (e.g.,decaying the step size too fast) for non-convex problems.

ADAGRAD

The update rule of the diagonal version of ADAGRAD in [82] is given as

$$\begin{aligned} G_n &= G_{n-1} + \operatorname{diag}(g_n \odot g_n) \\ \pi_{n+1} &= \arg\min_{\pi \in \Pi} \langle g_n, \pi \rangle + \frac{1}{2\eta}(\pi - \pi_n)^\top (\epsilon I + G_n)^{1/2}(\pi - \pi_n) \end{aligned} \tag{B.17}$$

where $G_0 = 0$ and $\eta > 0$ is a constant. ADAGRAD is designed to be optimal for online linear optimization problems. Above we provide the update equations of its mirror descent formulation in (B.17); a similar FTRL is also available (again the difference only happens when $\Pi$ is constrained).

In terms of our notation, its `update` and `project` are defined standardly as in (B.8), i.e.

$$\texttt{update}(h_n, H_n, g_n, w_n) = \arg\min_{\pi' \in \Pi} \langle w_n g_n, \pi' \rangle + B_{H_n}(\pi' || \pi_n) \tag{B.18}$$

and its `adapt` essentially only updates $G_n$:

$$\texttt{adapt}(h_n, H_{n-1}, g_n, w_n) : G_n = G_{n-1} + \operatorname{diag}(w_n g_n \odot w_n g_n)$$

where the regularization is defined the updated $G_n$ and the constant $\eta$ as

$$H_n(\pi) = \frac{1}{2\eta} \pi^\top (\epsilon I + G_n)^{1/2} \pi.$$

One can simply verify the above definitions of `update` and `adapt` agrees with (B.17).

## B.4 A Practical Variation of PICCOLO

In Section 4.4.2, we show that, given a base algorithm in mirror descent/FTRL, PICCOLO generates a new first-order update rule by recomposing the three basic operations into

$$h_n = \texttt{update}(\hat{h}_n, H_{n-1}, \hat{g}_n, w_n) \qquad \text{[Prediction]} \qquad \text{(B.19)}$$

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, H_{n-1}, e_n, w_n) \\ \hat{h}_{n+1} &= \texttt{update}(h_n, H_n, e_n, w_n) \end{aligned} \qquad \text{[Correction]} \qquad \text{(B.20)}$$

where $e_n = g_n - \hat{g}_n$ and $\hat{g}_n$ is an estimate of $g_n$ given by a predictive model $\Phi_n$.

Here we propose a slight variation which introduces another operation `shift` inside the Prediction Step. This leads to the new set of update rules:

$$\begin{aligned} \hat{H}_n &= \texttt{shift}(\hat{h}_n, H_{n-1}) \\ h_n &= \texttt{update}(\hat{h}_n, \hat{H}_n, \hat{g}_n, w_n) \end{aligned} \qquad \text{[Prediction]} \qquad \text{(B.21)}$$

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, \hat{H}_n, e_n, w_n) \\ \hat{h}_{n+1} &= \texttt{update}(h_n, H_n, e_n, w_n) \end{aligned} \qquad \text{[Correction]} \qquad \text{(B.22)}$$

The new `shift` operator additionally changes the regularization based on $\hat{h}_n$ the current representation of the policy in the Prediction Step, *independent* of the predicted gradient $\hat{g}_n$ and weight $w_n$. The main purpose of including this additional step is to deal with numerical difficulties, such as singularity of $H_n$. For example, in natural gradient descent, the Fisher information of some policy can be close to being singular along the direction of the gradients that are evaluated at different policies. As a result, in the original Prediction Step of PICCOLO, $H_{n-1}$ which is evaluated at $\pi_{n-1}$ might be singular in the direction of $\hat{g}_n$ which is evaluated $\hat{\pi}_n$.

The new operator `shift` brings in an extra degree of freedom to account for such issue. Although from a theoretical point of view (cf. Appendix B.6) the use of `shift` would

only increase regrets and should be avoided if possible, in practice, its merits in handling numerical difficulties can out weight the drawback. Because `shift` does not depend on the size of $\hat{g}_n$ and $w_n$, the extra regrets would only be proportional to $O(\sum_{n=1}^{N} \|\pi_n - \hat{\pi}_n\|_n)$, which can be smaller than other terms in the regret bound (see Appendix B.6).

### B.5 Example: PICCOLOing Natural Gradient Descent

We give an alternative example to illustrate how one can use the above procedure to "PICCOLO" a base algorithm into a new algorithm. Here we consider the adaptive natural gradient descent rule in Appendix B.3 as the base algorithm, which (given first-order information $g_n$ and weight $w_n$) updates the policy through

$$\pi_{n+1} = \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + \frac{\sqrt{\hat{G}_n}}{2\eta_n} (\pi - \pi_n)^\top F_n (\pi - \pi_n) \tag{B.23}$$

where $F_n$ is the Fisher information matrix of policy $\pi_n$ [176], $\eta_n$ a scheduled learning rate, and $\hat{G}_n$ is an adaptive multiplier for the step size which we will shortly describe. When $\hat{G}_n = 1$, the update in (B.23) gives the standard natural gradient descent update with step size $\eta_n$ [99] .

The role of $\hat{G}_n$ is to adaptively and *slowly* changes the step size to minimize $\sum_{n=1}^{N} \frac{\eta_n}{\sqrt{G_n}} \|g_n\|^2_{F_n,*}$, which plays an important part in the regret bound (see Section 4.5, Appendix B.6, and e.g. [96] for details). To this end, we update $\hat{G}_n$ by setting (with $G_0 = 0$)

$$G_n = \beta_2 G_{n-1} + (1 - \beta_2) \tfrac{1}{2} g_n^\top F_n^{-1} g_n, \qquad \hat{G}_n = G_n / (1 - \beta_2^n) \tag{B.24}$$

similar to the moving average update rule in ADAM, and update $\eta_n$ in the same way as in the basic mirror descent algorithm (e.g.,$\eta_n = O(1/\sqrt{n})$). As a result, this leads to a similar regret like ADAM with $\beta_1 = 0$, but in terms of a local norm specified by the Fisher information matrix.

Now, let's see how to PICCOLO the adaptive natural gradient descent rule above. First,

it is easy to see that the adaptive natural gradient descent rule is an instance of mirror descent (with $h_n = \pi_n$ and $H_n(g) = \frac{\sqrt{\hat{G}_n}}{2\eta_n} g^\top F_n g$), so the `update` and `project` operations are defined in the standard way, as in Section 4.4.2. The `adapt` operation updates the iteration counter $n$, the learning rate $\eta_n$, and updates $\hat{G}_n$ through (B.24).

To be more specific, let us explicitly write out the Prediction Step and the Correction Step of the PICCOLOed adaptive natural gradient descent rule in closed form as below: e.g.,if $\eta_n = \frac{1}{\sqrt{n}}$, then we can write them as

$$\text{[Prediction]} \qquad \pi_n = \arg\min_{\pi \in \Pi} \langle w_n \hat{g}_n, \pi \rangle + \frac{\sqrt{\hat{G}_{n-1}}}{2\eta_{n-1}} (\pi - \hat{\pi}_n)^\top F_{n-1} (\pi - \hat{\pi}_n)$$

$$\text{[Correction]} \qquad \begin{aligned} \eta_n &= 1/\sqrt{n} \\ G_n &= \beta_2 G_{n-1} + (1-\beta_2) \tfrac{1}{2} g_n^\top F_n^{-1} g_n \\ \hat{G}_n &= G_n / (1-\beta_2^n) \\ \hat{\pi}_{n+1} &= \arg\min_{\pi \in \Pi} \langle w_n e_n, \pi \rangle + \frac{\sqrt{\hat{G}_n}}{2\eta_n} (\pi - \pi_n)^\top F_n (\pi - \pi_n) \end{aligned}$$

## B.6 Regret Analysis of PICCOLO

The main idea of PICCOLO is to achieve optimal performance in predictable online learning problems by *reusing* existing adaptive, optimal first-order algorithms that are designed for adversarial online learning problems. This is realized by the reduction techniques presented in this section.

Here we prove the performance of PICCOLO in general predictable online learning problems, independent of the context of policy optimization. In Appendix B.6.1, we first show an elegant reduction from predictable problems to adversarial problems. Then we prove Theorem 1 in Appendix B.6.2, showing how the optimal regret bound for predictable linear problems can be achieved by PICCOLOing mirror descent and FTRL algorithms. Note that we will abuse the notation $l_n$ to denote the per-round losses in this general setting.

### B.6.1 Reduction from Predictable Online Learning to Adversarial Online Learning

Consider a predictable online learning problem with per-round losses $\{l_n\}$. Suppose in round $n$, before playing $\pi_n$ and revealing $l_n$, we have access to some prediction of $l_n$, called $\hat{l}_n$. In particular, we consider the case where $\hat{l}_n(\pi) = \langle \hat{g}_n, \pi \rangle$ for some vector $\hat{g}_n$. Running an (adaptive) online learning algorithm designed for the general adversarial setting is not optimal here, as its regret would be in $O(\sum_{n=1}^{N} \|\nabla l_n\|_{n,*}^2)$, where $\|\cdot\|_n$ is some local norm chosen by the algorithm and $\|\cdot\|_{n,*}$ is its dual norm. Ideally, we would only want to pay for the information that is unpredictable. Specifically, we wish to achieve an optimal regret in $O(\sum_{n=1}^{N} \|\nabla l_n - \nabla \hat{l}_n\|_{n,*}^2)$ instead [81].

To achieve the optimal regret bound yet without referring to specialized, nested two-step algorithms (e.g.,mirror-prox [91], optimistic mirror descent [171], FTRL-prediction [81]), we consider decomposing a *predictable* problem with $N$ rounds into an *adversarial* problem with $2N$ rounds:

$$\sum_{n=1}^{N} l_n(\pi_n) = \sum_{n=1}^{N} \hat{l}_n(\pi_n) + \delta_n(\pi_n) \tag{B.25}$$

where $\delta_n = l_n - \hat{l}_n$. Therefore, we can treat the predictable problem as a new adversarial online learning problem with a loss sequence $\hat{l}_1, \delta_1, \hat{l}_2, \delta_2, \ldots, \hat{l}_N, \delta_N$ and consider solving this new problem with some standard online learning algorithm designed for the adversarial setting.

Before analysis, we first introduce a new decision variable $\hat{\pi}_n$ and denote the decision sequence in this new problem as $\hat{\pi}_1, \pi_1, \hat{\pi}_2, \pi_2, \ldots, \hat{\pi}_N, \pi_N$, so the definition of the variables are consistent with that in the problem before. Because this new problem is unpredictable,

the optimal regret of this new decision sequence is

$$\sum_{n=1}^{N} \hat{l}_n(\hat{\pi}_n) + \delta_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{N} \hat{l}_n(\pi) + \delta_n(\pi) = O(\sum_{n=1}^{N} \|\nabla \hat{l}_n\|_{n,*}^2 + \|\nabla \delta_n\|_{n+1/2,*}^2) \tag{B.26}$$

where the subscript $n + 1/2$ denotes the extra round due to the reduction.

At first glance, our reduction does not meet the expectation of achieving regret in $O(\sum_{n=1}^{N} \|\nabla l_n - \nabla \hat{l}_n\|_{n,*}^2) = O(\sum_{n=1}^{N} \|\nabla \delta_n\|_{n,*}^2)$. However, we note that the regret for the new problem is too loose for the regret of the original problem, which is

$$\sum_{n=1}^{N} \hat{l}_n(\pi_n) + \delta_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{N} \hat{l}_n(\pi) + \delta_n(\pi)$$

where the main difference is that originally we care about $\hat{l}_n(\pi_n)$ rather than $\hat{l}_n(\hat{\pi}_n)$. Specifically, we can write

$$\begin{aligned} \sum_{n=1}^{N} l_n(\pi_n) &= \sum_{n=1}^{N} \hat{l}_n(\pi_n) + \delta_n(\pi_n) \\ &= \left( \sum_{n=1}^{N} \hat{l}_n(\hat{\pi}_n) + \delta_n(\pi_n) \right) + \left( \sum_{n=1}^{N} \hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n) \right) \end{aligned}$$

Therefore, if the update rule for generating the decision sequence $\hat{\pi}_1, \pi_1, \hat{\pi}_2, \pi_2, \ldots, \hat{\pi}_N, \pi_N$ contributes sufficient negativity in the term $\hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)$ compared with the regret of the new adversarial problem, then the regret of the original problem can be smaller than (B.26). This is potentially possible, as $\pi_n$ is made after $\hat{l}_n$ is revealed. Especially, in the fixed-point formulation of PICCOLO, $\pi_n$ and $\hat{l}_n$ can be decided simultaneously.

In the next section, we show that when the base algorithm, which is adopted to solve the new adversarial problem given by the reduction, is in the family of mirror descent and FTRL. Then the regret bound of PICCOLO with respect to the original predictable problem is optimal.

### B.6.2 Optimal Regret Bounds for Predictable Problems

We show that if the base algorithm of PICCOLO belongs to the family of optimal mirror descent and FTRL designed for adversarial problems, then PICCOLO can achieve the optimal regret of predictable problems. In this subsection, we assume the loss sequence is linear, i.e.,$l_n(\pi) = \langle g_n, \pi \rangle$ for some $g_n$, and the results are summarized as Theorem 1 in the main text (in a slightly different notation).

*Mirror Descent*

First, we consider mirror descent as the base algorithm. In this case, we can write the PICCOLO update rule as

$$\pi_n = \arg\min_{\pi \in \Pi} \left\langle \nabla \hat{l}_n(\hat{\pi}_n), x \right\rangle + B_{H_{n-1}}(\pi || \hat{\pi}_n) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \arg\min_{\pi \in \Pi} \langle \nabla \delta_n(\pi_n), \pi \rangle + B_{H_n}(\pi || \pi_n) \qquad \text{[Correction]}$$

where $H_n$ can be updated based on $e_n := \nabla \delta_n(\pi_n) = \nabla l_n(\pi_n) - \nabla \hat{l}_n(\hat{\pi}_n)$ (recall by definition $\nabla l_n(\pi_n) = g_n$ and $\nabla \hat{l}_n(\hat{\pi}_n) = \nabla \hat{l}_n(\pi_n) = \hat{g}_n$). Notice that in the Prediction Step, PICCOLO uses the regularization from the previous Correction Step.

To analyze the performance, we use a lemma of the mirror descent's properties. The proof is a straightforward application of the optimality condition of the proximal map [179]. We provide a proof here for completeness.

**Lemma 5.** *Let $\mathcal{K}$ be a convex set. Suppose $R$ is 1-strongly convex with respect to norm $\|\cdot\|$. Let $g$ be a vector in some Euclidean space and let*

$$y = \arg\min_{z \in \mathcal{K}} \langle g, z \rangle + \frac{1}{\eta} B_R(z || x)$$

*Then for all* $z \in \mathcal{K}$

$$\eta \langle g, y - z \rangle \leq B_R(z||x) - B_R(z||y) - B_R(y||x) \tag{B.27}$$

*which implies*

$$\eta \langle g, x - z \rangle \leq B_R(z||x) - B_R(z||y) + \frac{\eta^2}{2}\|g\|_*^2 \tag{B.28}$$

*Proof.* Recall the definition $B_R(z||x) = R(z) - R(x) - \langle \nabla R(x), z - x \rangle$. The optimality of the proximal map can be written as

$$\langle \eta g + \nabla R(y) - \nabla R(x), y - z \rangle \leq 0, \qquad \forall z \in \mathcal{K}$$

By rearranging the terms, we can rewrite the above inequality in terms Bregman divergences as follows and derive the first inequality (B.27):

$$\begin{aligned}
\langle \eta g, y - z \rangle &\leq \langle \nabla R(x) - \nabla R(y), y - z \rangle \\
&= B_R(z||x) - B_R(z||y) + \langle \nabla R(x) - \nabla R(y), y \rangle - \langle \nabla R(x), x \rangle + \langle \nabla R(y), y \rangle \\
&\quad + R(x) - R(y) \\
&= B_R(z||x) - B_R(z||y) + \langle \nabla R(x), y - x \rangle + R(x) - R(y) \\
&= B_R(z||x) - B_R(z||y) - B_R(y||x)
\end{aligned}$$

The second inequality is the consequence of (B.27). First, we rewrite (B.27) as

$$\langle \eta g, x - z \rangle = B_R(z||x) - B_R(z||y) - B_R(y||x) + \langle \eta g, x - y \rangle$$

Then we use the fact that $B_R$ is 1-strongly convex with respect to $\|\cdot\|$, which implies

$$-B_R(y||x) + \langle \eta g, x - y \rangle \leq -\frac{1}{2}\|x - y\|^2 + \langle \eta g, x - y \rangle \leq \frac{\eta^2}{2}\|g\|_*^2$$

Combining the two inequalities yields (B.28). □

Lemma 5 is usually stated with (B.28), which concerns the decision made before seeing the per-round loss (as in the standard adversarial online learning setting). Here, we additionally concern $\hat{l}_n(\pi_n)$, which is the decision made after seeing $\hat{l}_n$, so we need a tighter bound (B.27).

Now we show that the regret bound of PICCOLO in the predictable linear problems when the base algorithm is mirror descent.

**Proposition 3.** *Assume the base algorithm of* PICCOLO *is mirror descent satisfying Assumption 1. Let* $g_n = \nabla l_n(\pi_n)$ *and* $e_n = g_n - \hat{g}_n$. *Then it holds that, for any* $\pi \in \Pi$,

$$\sum_{n=1}^{N} w_n \langle g_n, \pi_n - \pi \rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2}\|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$$

*Proof.* Suppose $R_n$, which is defined by $H_n$, is 1-strongly convex with respect to $\|\cdot\|_n$. Then by Lemma 5, we can write, for all $\pi \in \Pi$,

$$\begin{aligned} w_n \langle g_n, \pi_n - \pi \rangle &= w_n \langle \hat{g}_n, \pi_n - \pi \rangle + w_n \langle e_n, \pi_n - \pi \rangle \\ &\leq B_{R_{n-1}}(\pi||\hat{\pi}_n) - B_{R_{n-1}}(\pi||\pi_n) - B_{R_{n-1}}(\pi_n||\hat{\pi}_n) \\ &\quad + B_{R_n}(\pi||\pi_n) - B_{R_n}(\pi||\hat{\pi}_{n+1}) + \frac{w_n^2}{2}\|e_n\|_{*,n}^2 \end{aligned} \tag{B.29}$$

where we use (B.27) for $\hat{g}_n$ and (B.28) for the loss $e_n$.

To show the regret bound of the original (predictable) problem, we first notice that

$$\begin{aligned}
&\sum_{n=1}^{N} B_{R_{n-1}}(\pi||\hat{\pi}_n) - B_{R_{n-1}}(\pi||\pi_n) + B_{R_n}(\pi||\pi_n) - B_{R_n}(\pi||\hat{\pi}_{n+1}) \\
&= B_{R_0}(\pi||\hat{\pi}_1) - B_{R_N}(\pi||\hat{\pi}_{N+1}) + \\
&\quad \sum_{n=1}^{N} B_{R_{n-1}}(\pi||\hat{\pi}_n) - B_{R_{n-1}}(\pi||\pi_n) + B_{R_n}(\pi||\pi_n) - B_{R_{n-1}}(\pi||\hat{\pi}_n) \\
&= B_{R_0}(\pi||\hat{\pi}_1) - B_{R_N}(\pi||\hat{\pi}_{N+1}) + \sum_{n=1}^{N} B_{R_n}(\pi||\pi_n) - B_{R_{n-1}}(\pi||\pi_n) \leq M_N
\end{aligned}$$

where the last inequality follows from the assumption on the base algorithm. Therefore, by telescoping the inequality in (B.29) and using the strong convexity of $R_n$, we get

$$\begin{aligned}
\sum_{n=1}^{N} w_n \langle g_n, \pi_n - \pi \rangle &\leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - B_{R_{n-1}}(\pi_n||\hat{\pi}_n) \\
&\leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2 \qquad \square
\end{aligned}$$

*Follow-the-Regularized-Leader*

We consider another type of base algorithm, FTRL, which is mainly different from mirror descent in the way that constrained decision sets are handled [96]. In this case, the exact update rule of PICCOLO can be written as

$$\pi_n = \arg\min_{\pi \in \Pi} \langle w_n \hat{g}_n, \pi \rangle + \sum_{m=1}^{n-1} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi||\pi_m) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \arg\min_{\pi \in \Pi} \sum_{m=1}^{n} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi||\pi_m) \qquad \text{[Correction]}$$

From the above equations, we verify that MOBIL [9] is indeed a special case of PICCOLO, when the base algorithm is FTRL.

We show PICCOLO with FTRL has the following guarantee.

**Proposition 4.** *Assume the base algorithm of* PICCOLO *is FTRL satisfying the Assumption 1. Then it holds that, for any* $\pi \in \Pi$*,*

$$\sum_{n=1}^{N} w_n \langle g_n, \pi_n - \pi \rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - \frac{1}{2} \|\pi_n - \hat{\pi}_n\|_{n-1}^2$$

We show the above results of PICCOLO using a different technique from [9]. Instead of developing a specialized proof like they do, we simply use the properties of FTRL on the $2N$-step new adversarial problem!

To do so, we recall some facts of the base algorithm FTRL. First, FTRL in (B.11) is equivalent to Follow-the-Leader (FTL) on a surrogate problem with the per-round loss is $\langle g_n, \pi \rangle + B_{r_n}(\pi || \pi_n)$. Therefore, the regret of FTRL can be bounded by the regret of FTL in the surrogate problem plus the size of the additional regularization $B_{r_n}(\pi || \pi_n)$. Second, we recall a standard techniques in proving FTL, called Strong FTL Lemma (see e.g. [96]), which is proposed for *adversarial* online learning.

**Lemma 6** (Strong FTL Lemma [96])**.** *For any sequence* $\{\pi_n \in \Pi\}$ *and* $\{l_n\}$*,*

$$\mathrm{Regret}_N(l) := \sum_{n=1}^{N} l_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{N} l_n(\pi) \leq \sum_{n=1}^{N} l_{1:n}(\pi_n) - l_{1:n}(\pi_n^\star)$$

*where* $\pi_n^\star \in \arg\min_{\pi \in \Pi} l_{1:n}(\pi)$.

Using the decomposition idea above, we show the performance of PICCOLO following sketch below: first, we show a bound on the regret in the surrogate predictable problem with per-round loss $\langle g_n, \pi \rangle + B_{r_n}(\pi || \pi_n)$; second, we derive the bound for the original predictable problem with per-round loss $\langle g_n, \pi \rangle$ by considering the effects of $B_{r_n}(\pi || \pi_n)$. We will prove the first step by applying FTL on the transformed $2N$-step adversarial problem of the original $N$-step predictable surrogate problem and then showing that PICCOLO achieves the optimal regret in the original $N$-step predictable surrogate problem.

**Lemma 7** (Stronger FTL Lemma [9])**.** *For any sequence* $\{\pi_n\}$ *and* $\{l_n\}$*,*

$$\mathrm{Regret}_N(l) = \sum_{n=1}^{N} l_{1:n}(\pi_n) - l_{1:n}(\pi_n^\star) - \Delta_n$$

*where* $\Delta_{n+1} := l_{1:n}(\pi_{n+1}) - l_{1:n}(\pi_n^\star) \geq 0$ *and* $\pi_n^\star \in \arg\min_{\pi\in\Pi} l_{1:n}(\pi)$*.*

Our new reduction-based regret bound is presented below.

**Proposition 5.** *Let* $\{l_n\}$ *be a predictable loss sequence with predictable information* $\{\hat{l}_n\}$*. Suppose the decision sequence* $\hat{\pi}_1, \pi_1, \hat{\pi}_2, \ldots, \hat{\pi}_N, \pi_N$ *is generated by running FTL on the transformed adversarial loss sequence* $\hat{l}_1, \delta_1, \hat{l}_2, \ldots, \hat{l}_N, \delta_N$*, then the bound in the Stronger FTL Lemma holds. That is,* $\mathrm{Regret}_N(l) \leq \sum_{n=1}^{N} l_{1:n}(\pi_n) - l_{1:n}(\pi_n^\star) - \Delta_n$*, where* $\Delta_{n+1} := l_{1:n}(\pi_{n+1}) - l_{1:n}(\pi_n^\star) \geq 0$ *and* $\pi_n^\star \in \arg\min_{\pi\in\Pi} l_{1:n}(\pi)$*.*

*Proof.* First, we transform the loss sequence and write

$$\sum_{n=1}^{N} l_n(\pi_n) = \sum_{n=1}^{N} \hat{l}_n(\pi_n) + \delta_n(\pi_n) = \left(\sum_{n=1}^{N} \hat{l}_n(\hat{\pi}_n) + \delta_n(\pi_n)\right) + \left(\sum_{n=1}^{N} \hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)\right)$$

Then we apply standard Strong FTL Lemma on the new adversarial problem in the left term.

$$\begin{aligned}
&\sum_{n=1}^{N} \hat{l}_n(\hat{\pi}_n) + \delta_n(\pi_n) \\
&\leq \sum_{n=1}^{N} (\hat{l} + \delta)_{1:n}(\pi_n) - \min_{\pi\in\Pi} (\hat{l} + \delta)_{1:n}(\pi) \\
&\quad + \sum_{n=1}^{N} ((\hat{l} + \delta)_{1:n-1} + \hat{l}_n)(\hat{\pi}_n) - \min_{\pi\in\Pi} ((\hat{l} + \delta)_{1:n-1} + \hat{l}_n)(\pi) \\
&= \sum_{n=1}^{N} l_{1:n}(\pi_n) - \min_{\pi\in\Pi} l_{1:n}(\pi) + \sum_{n=1}^{N} (l_{1:n-1} + \hat{l}_n)(\hat{\pi}_n) - (l_{1:n-1} + \hat{l}_n)(\pi_n)
\end{aligned}$$

where the first inequality is due to Strong FTL Lemma and the second equality is because FTL update assumption.

Now we observe that if we add the second term above and $\sum_{n=1}^{N} \hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)$ together,

we have

$$\sum_{n=1}^{N}(l_{1:n-1}+\hat{l}_n)(\hat{\pi}_n)-(l_{1:n-1}+\hat{l}_n)(\pi_n)+(\hat{l}_n(\pi_n)-\hat{l}_n(\hat{\pi}_n))$$
$$=\sum_{n=1}^{N}(l_{1:n-1})(\hat{\pi}_n)-l_{1:n-1}(\pi_n)=\Delta_n$$

Thus, combing previous two inequalities, we have the bound in the Stronger FTL Lemma:

$$\sum_{n=1}^{N}l_n(\pi_n)\leq\sum_{n=1}^{N}l_{1:n}(\pi_n)-\min_{\pi\in\Pi}l_{1:n}(\pi)-\Delta_n \qquad \square$$

Using Proposition 5, we can now bound the regret of PICCOLO in Proposition 4 easily.

*Proof of Proposition 4.* Suppose $\sum_{m=1}^{n}B_{r_m}(\cdot||\pi_m)$ is 1-strongly convex with respect to some norm $\|\cdot\|_n$. Let $f_n=\langle w_n g_n,\pi_n\rangle+B_{r_n}(\pi||\pi_m)$. Then by a simple convexity analysis (see e.g.,see [96]) and Proposition 5, we can derive

$$\begin{aligned}\mathrm{Regret}_N(f)&\leq\sum_{n=1}^{N}(f_{1:n}(\pi_n)-\min_{\pi\in\Pi}f_{1:n}(\pi))-(f_{1:n-1}(\pi_n)-f_{1:n-1}(\hat{\pi}_n))\\&\leq\sum_{n=1}^{N}\frac{w_n^2}{2}\|e_n\|_{n,*}^2-\frac{1}{2}\|\pi_n-\hat{\pi}_n\|_{n-1}^2\end{aligned}$$

Finally, because $r_n$ is proximal (i.e.,$B_{r_n}(\pi_n||\pi_n)=0$), we can bound the original regret: for any $\pi\in\Pi$, it satisfies that

$$\begin{aligned}\sum_{n=1}^{N}w_n\langle g_n,\pi_n-\pi\rangle&\leq\sum_{n=1}^{N}f_n(\pi_n)-f_n(\pi)+B_{r_n}(\pi||\pi_n)\\&\leq M_N+\sum_{n=1}^{N}\frac{w_n^2}{2}\|e_n\|_{*,n}^2-\frac{1}{2}\|\pi_n-\hat{\pi}_n\|_{n-1}^2\end{aligned}$$

where we use Assumption 1 and the bound of $\mathrm{Regret}_N(f)$ in the second inequality. $\square$

## B.7 Policy Optimization Analysis of PICCOLO

In this section, we discuss how to interpret the bound given in Theorem 1

$$\sum_{n=1}^{N} w_n \langle g_n, \pi_n - \pi \rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - \frac{1}{2} \|\pi_n - \hat{\pi}_n\|_{n-1}^2$$

in the context of policy optimization and show exactly how the optimal bound

$$\mathbb{E}\left[\sum_{n=1}^{N} \langle w_n g_n, \pi_n - \pi \rangle\right] \leq O(1) + C_{\Pi,\Phi} \frac{w_{1:N}}{\sqrt{N}} \tag{B.30}$$

is derived. We will discuss how model learning can further help minimize the regret bound later in Appendix B.7.4.

### B.7.1 Assumptions

We introduce some assumptions to characterize the sampled gradient $g_n$. Recall $g_n = \nabla \tilde{l}_n(\pi_n)$.

**Assumption 2.** $\|\mathbb{E}[g_n]\|_*^2 \leq G_g^2$ and $\|g_n - \mathbb{E}[g_n]\|_*^2 \leq \sigma_g^2$ for some finite constants $G_g$ and $\sigma_g$.

Similarly, we consider properties of the predictive model $\Phi_n$ that is used to estimate the gradient of the next per-round loss. Let $\mathcal{P}$ denote the class of these models (i.e.,$\Phi_n \in \mathcal{P}$), which can potentially be *stochastic*. We make assumptions on the size of $\hat{g}_n$ and its variance.

**Assumption 3.** $\|\mathbb{E}[\hat{g}_n]\|_*^2 \leq G_{\hat{g}}^2$ and $\mathbb{E}[\|\hat{g}_n - \mathbb{E}[\hat{g}_n]\|_*^2] \leq \sigma_{\hat{g}}^2$ for some finite constants $G_{\hat{g}}$ and $\sigma_{\hat{g}}$.

Additionally, we assume these models are Lipschitz continuous.

**Assumption 4.** There is a constant $L \in [0, \infty)$ such that, for any instantaneous cost $\psi$ and any $\Phi \in \mathcal{P}$, it satisfies $\|\mathbb{E}[\Phi(\pi)] - \mathbb{E}[\Phi(\pi')]\|_* \leq L\|\pi - \pi'\|$.

Lastly, as PICCOLO is agnostic to the base algorithm, we assume the local norm $\|\cdot\|_n$ chosen by the base algorithm at round $n$ satisfies $\|\cdot\|_n^2 \geq \alpha_n \|\cdot\|^2$ for some $\alpha_n > 0$. This condition implies that $\|\cdot\|_{n,*}^2 \leq \frac{1}{\alpha_n}\|\cdot\|_*^2$. In addition, we assume $\alpha_n$ is non-decreasing so that $M_N = O(\alpha_N)$ in Assumption 1, where the leading constant in the bound $O(\alpha_N)$ is proportional to $|\Pi|$, as commonly chosen in online convex optimization.

### B.7.2 A Useful Lemma

We study the bound in Theorem 1 under the assumptions made in the previous section. We first derive a basic inequality, following the idea in [9, Lemma 4.3].

**Lemma 8.** *Under Assumptions 2, 3, and 4, it holds*

$$\mathbb{E}[\|e_n\|_{*,n}^2] = \mathbb{E}[\|g_n - \hat{g}_n\|_{*,n}^2] \leq \frac{4}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + L_n^2\|\pi_n - \hat{\pi}_n\|_n^2 + E_n(\Phi_n)\right)$$

*where* $E_n(\Phi_n) = \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_*^2$ *is the prediction error of model* $\Phi_n$.

*Proof.* Recall $\hat{g}_n = \Phi_n(\hat{\pi}_n, \psi_n)$. Using the triangular inequality, we can simply derive

$$\begin{aligned}
&\mathbb{E}[\|g_n - \hat{g}_n\|_{*,n}^2] \\
&\leq 4\left(\mathbb{E}[\|g_n - \mathbb{E}[g_n]\|_{*,n}^2] + \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_{*,n}^2\right) + \\
&\quad 4\left(\|\mathbb{E}[\Phi_n(\pi_n, \psi_n)] - \mathbb{E}[\hat{g}_n]\|_{*,n}^2 + \mathbb{E}[\|\mathbb{E}[\hat{g}_n] - \hat{g}_n\|_{*,n}^2]\right) \\
&= 4\left(\mathbb{E}[\|g_n - \mathbb{E}[g_n]\|_{*,n}^2] + \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_{*,n}^2\right) + \\
&\quad 4\left(\|\mathbb{E}[\Phi_n(\pi_n, \psi_n)] - \mathbb{E}[\Phi_n(\hat{\pi}_n, \psi_n)]\|_{*,n}^2 + \mathbb{E}[\|\mathbb{E}[\hat{g}_n] - \hat{g}_n\|_{*,n}^2]\right) \\
&\leq 4\left(\frac{1}{\alpha_n}\sigma_g^2 + \frac{1}{\alpha_n}E_n(\Phi_n) + \|\mathbb{E}[\Phi_n(\pi_n, \psi_n)] - \mathbb{E}[\Phi_n(\hat{\pi}_n, \psi_n)]\|_{*,n}^2 + \frac{1}{\alpha_n}\sigma_{\hat{g}}^2\right) \\
&\leq \frac{4}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + L^2\|\pi_n - \hat{\pi}_n\|_n^2 + E_n(\Phi_n)\right)
\end{aligned}$$

where the last inequality is due to Assumption 4. □

### B.7.3 Optimal Regret Bounds

We now analyze the regret bound in Theorem 1

$$\sum_{n=1}^{N} w_n \langle g_n, \pi_n - \pi \rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - \frac{1}{2} \|\pi_n - \hat{\pi}_n\|_{n-1}^2 \tag{B.31}$$

We first gain some intuition about the size of

$$M_N + \mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2\right]. \tag{B.32}$$

Because when $\texttt{adapt}(h_n, H_{n-1}, e_n, w_n)$ is called in the Correction Step in (B.20) with the error gradient $e_n$ as input, an optimal base algorithm (e.g.,all the base algorithms listed in Appendix B.3) would choose a local norm sequence $\|\cdot\|_n$ such that (B.32) is optimal. For example, suppose $\|e_n\|_*^2 = O(1)$ and $w_n = n^p$ for some $p > -1$. If the base algorithm is basic mirror descent (cf. Appendix B.3), then $\alpha_n = O(\frac{w_{1:n}}{\sqrt{n}})$. By our assumption that $M_N = O(\alpha_N)$, it implies (B.32) can be upper bounded by

$$\begin{aligned} M_N + \mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n^2}{2} \|e_n\|_{*,n}^2\right] &\leq O\left(\frac{w_{1:N}}{\sqrt{N}}\right) + \left[\sum_{n=1}^{N} \frac{w_n^2 \sqrt{n}}{2 w_{1:n}} \|e_n\|_*^2\right] \\ &\leq O\left(\frac{w_{1:N}}{\sqrt{N}} + \sum_{n=1}^{N} \frac{w_n^2 \sqrt{n}}{w_{1:n}}\right) = O\left(N^{p+1/2}\right) \end{aligned}$$

which will lead to an optimal weighted average regret in $O(\frac{1}{\sqrt{N}})$.

PICCOLO actually has a better regret than the simplified case discussed above, because of the negative term $-\frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$ in (B.31). To see its effects, we combine Lemma 8

with (B.31) to reveal some insights:

$$\mathbb{E}\left[\sum_{n=1}^{N} w_n \left\langle g_n, \pi_n - \pi \right\rangle\right] \tag{B.33}$$

$$\leq O(\alpha_N) + \mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n^2}{2}\|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2\right] \tag{B.34}$$

$$\leq O(\alpha_N) + \mathbb{E}\left[\sum_{n=1}^{N} \frac{2w_n^2}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + L^2\|\pi_n - \hat{\pi}_n\|_n^2 + E_n(\Phi_n)\right) - \frac{\alpha_{n-1}}{2}\|\pi_n - \hat{\pi}_n\|^2\right]$$

$$= \left(O(\alpha_N) + \mathbb{E}\left[\sum_{n=1}^{N} \frac{2w_n^2}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + E_n(\Phi_n)\right)\right]\right) + \left(\mathbb{E}\left[\sum_{n=1}^{N}(\frac{2w_n^2}{\alpha_n}L^2 - \frac{\alpha_{n-1}}{2})\|\pi_n - \hat{\pi}_n\|^2\right]\right) \tag{B.35}$$

The first term in (B.35) plays the same role as (B.32); when the base algorithm has an optimal `adapt` operation and $w_n = n^p$ for some $p > -1$, it would be in $O\left(N^{p+1/2}\right)$. Here we see that the constant factor in this bound is proportional to $\sigma_g^2 + \sigma_{\hat{g}}^2 + E_n(\Phi_n)$. Therefore, if the variances $\sigma_g^2$, $\sigma_{\hat{g}}^2$ of the gradients are small, the regret would mainly depend on the prediction error $E_n(\Phi_n)$ of $\Phi_n$. In the next section (Appendix B.7.4), we will show that when $\Phi_n$ is learned online (as the authors in [9] suggest), on average the regret is close to the regret of using the best model in the hindsight. The second term in (B.35) contributes to $O(1)$ in the regret, when the base algorithm adapts properly to $w_n$. For example, when $\alpha_n = \Theta(\frac{w_{1:n}}{\sqrt{n}})$ and $w_n = n^p$ for some $p > -1$, then

$$\sum_{n=1}^{N} \frac{2w_n^2}{\alpha_n}L^2 - \frac{\alpha_{n-1}}{2} = \sum_{n=1}^{N} O(n^{p-1/2} - n^{p+1/2}) = O(1)$$

In addition, because $\|\pi_n - \hat{\pi}_n\|$ would converge to zero, the effects of the second term in (B.35) becomes even minor.

In summary, for a reasonable base algorithm and $w_n = n^p$ with $p > -1$, running

PICCOLO has the regret bound

$$\mathbb{E}\left[\sum_{n=1}^{N} w_n \left\langle g_n, \pi_n - \pi \right\rangle\right] = O(\alpha_N) + O\left(\frac{w_{1:N}}{\sqrt{N}}(\sigma_g^2 + \sigma_{\hat{g}}^2)\right) + O(1) + \mathbb{E}\left[\sum_{n=1}^{N} \frac{2w_n^2}{\alpha_n} E_n(\Phi_n)\right] \tag{B.36}$$

Suppose $\alpha_n = \Theta(|\Pi|\frac{w_{1:n}}{\sqrt{n}})$ and $w_n = n^p$ for some $p > -1$, This implies the inequality

$$\mathbb{E}\left[\sum_{n=1}^{N} \left\langle w_n g_n, \pi_n - \pi \right\rangle\right] \leq O(1) + C_{\Pi,\Phi}\frac{w_{1:N}}{\sqrt{N}} \tag{(B.30)}$$

where $C_{\Pi,\Phi} = O(|\Pi| + \sigma_g^2 + \sigma_{\hat{g}}^2 + \sup_n E_n(\Phi_n))$. The use of non-uniform weights can lead to a faster on average decay of the standing $O(1)$ term in the final weighted average regret bound, i.e.

$$\frac{1}{w_{1:N}}\mathbb{E}\left[\sum_{n=1}^{N} \left\langle w_n g_n, \pi_n - \pi \right\rangle\right] \leq O\left(\frac{1}{w_{1:N}}\right) + \frac{C_{\Pi,\Phi}}{\sqrt{N}}$$

In general, the authors in [86, 9] recommend using $p \ll N$ (e.g.,in the range of $[0, 5]$) to remove the undesirable constant factor, yet without introducing large multiplicative constant factor.

### B.7.4 Model Learning

The regret bound in (B.36) reveals an important factor that is due to the prediction error $\mathbb{E}\left[\sum_{n=1}^{N} \frac{2w_n^2}{\alpha_n} E_n(\Phi_n)\right]$, where we recall $E_n(\Phi_n) = \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n)]\|_*^2$. To minimize this error sum through model learning, a secondary online learning problem with per-round loss $E_n(\cdot)$ can be considered [9]. Note that this is a standard weighted adversarial online learning problem (weighted by $\frac{2w_n^2}{\alpha_n}$), because $E_n(\cdot)$ is revealed after one commits to using model $\Phi_n$.

While in implementation the exact function $E_n(\cdot)$ is unavailable (as it requires infinite data), we can adopt an unbiased upper bound. For example, $E_n(\cdot)$ can be upper bounded by

the single- or multi-step prediction error of a transition dynamics model [9]. More generally, we can learn a neural network to minimize the gradient prediction error directly. As long as this secondary online learning problem is solved by a no-regret algorithm, the error due to online model learning would contribute a term in $O(w_{1:N}\epsilon_{\mathcal{P},N}/\sqrt{N}) + o(w_{1:N}/\sqrt{N})$ in (B.36), where $\epsilon_{\mathcal{P},N}$ is the minimal error achieved by the best model in the model class $\mathcal{P}$ (see [9] for details).

## B.8 Experimental Details

### B.8.1 Algorithms

**Base Algorithms** In the experiments, we consider three commonly used first-order online learning algorithms: ADAM, NATGRAD, and TRPO, all of which adapt the regularization online to alleviate the burden of learning rate tuning. We provide the decomposition of ADAM into the basic three operations in Appendix B.3, and that of NATGRAD in Appendix B.5. In particular, the adaptivity of NATGRAD is achieved by adjusting the step size based on a moving average of the dual norm of the gradient. TRPO adjusts the step size to minimize a given cost function (here it is a linear function defined by the first-order oracle) within a pre-specified KL divergence centered at the current decision. While greedily changing the step size in every iteration makes TRPO an inappropriate candidate for adversarial online learning. Nonetheless, it can still be written in the form of mirror descent and allows a decomposition using the three basic operators; its `adapt` operator can be defined as the process of finding the maximal scalar step along the natural gradient direction such that the updated decision stays within the trust region. For all the algorithms, a decaying step size multiplier in the form $\eta/(1 + \alpha\sqrt{n})$ is also used; for TRPO, it is used to specify the size of trust regions. The values chosen for the hyperparameters $\eta$ and $\alpha$ can be found in Table B.1. To the best of our knowledge, the conversion of these approaches into unbiased model-based algorithms is novel.

**Reinforcement Learning Per-round Loss** In iteration $n$, in order to compute the online gradient (4.5), GAE [101] is used to estimate the advantage function $A_{\pi_{n-1}}$. More concretely, this advantage estimate utilizes an estimate of value function $V_{\pi_{n-1}}$ (which we denote $\hat{V}_{\pi_{n-1}}$) and on-policy samples. We chosen $\lambda = 0.98$ in GAE to reduce influence of the error in $V_{\pi_{n-1}}$, which can be catastrophic. Importance sampling can be used to estimate $A_{\pi_{n-1}}$ in order to leverage data that are collected on-policy by running $\pi_n$. However, since we select a large $\lambda$, importance sampling can lead to vanishing importance weights, making the gradient extremely noisy. Therefore, in the experiments, importance sampling is not applied.

**Gradient Computation and Control Variate** The gradients are computed using likelihood-ratio trick and the associated advantage function estimates described above. A scalar control variate is further used to reduce the variance of the sampled gradient, which is set to the mean of the advantage estimates evaluated on newly collected data.

**Policies and Value Networks** Simple feed-forward neural networks are used to construct all of the function approximators (policy and value function) in the tasks. They have 1 hidden layer with 32 $\tanh$ units for all policy networks, and have 2 hidden layers with 64 $\tanh$ units for value function networks. Gaussian stochastic policies are considered, i.e., for any state $s \in \mathbb{S}$, $\pi_s$ is Gaussian, and the mean of $\pi_s$ is modeled by the policy network, whereas the diagonal covariance matrix is state independent (which is also learned). Initial value of $\log \sigma$ of the Gaussian policies $-1.0$, the standard deviation for initializing the output layer is $0.01$, and the standard deviation for initialization hidden layer is $1.0$. After the policy update, a new value function estimate $\hat{V}_{\pi_n}$ is computed by minimizing the mean of squared difference between $\hat{V}_{\pi_n}$ and $\hat{V}_{\pi_{n-1}} + \hat{A}_{\pi_n}$, where $\hat{A}_{\pi_n}$ is the GAE estimate using $\hat{V}_{\pi_{n-1}}$ and $\lambda = 0.98$, through ADAM with batch size 128, number of batches 2048, and learning rate 0.001. Value function is pretrained using examples collected by executing the randomly initialized policy.

**Computing Model Gradients** We compute $\hat{g}_n$ in two ways. The first approach is to use the simple heuristic that sets $\hat{g}_n = \Phi_n(\hat{\pi}_n)$, where $\Phi_n$ is some predictive models depending on the exact experimental setup. The second approach is to use the fixed-point formulation (4.8). This is realized by solving the equivalent optimization problem mentioned in the main text. In implementation, we only solves this problem approximately using some finite number of gradient steps; though this is insufficient to yield a stationary point as desired in the theory, we experimentally find that it is sufficient to yield improvement over the heuristic $\hat{g}_n = \Phi_n(\hat{\pi}_n)$.

**Approximate Solution to Fixed-Point Problems of PICCOLO** PICCOLO relies on the predicted gradient $\hat{g}_n$ in the Prediction Step. Recall ideally we wish to solve the fixed-point problem that finds $h_n^*$ such that

$$h_n^* = \texttt{update}(\hat{h}_n, H_{n-1}, \Phi_n(\pi_n(h_n^*)), w_n) \tag{B.37}$$

and then apply $\hat{g}_n = \Phi_n(\pi_n(h_n^*))$ in the Prediction Step to get $h_n$, i.e.,

$$h_n = \texttt{update}(\hat{h}_n, H_{n-1}, \hat{g}_n, w_n)$$

Because $h_n^*$ is the solution to the fixed-point problem, we have $h_n = h_n^*$. Such choice of $\hat{g}_n$ will fully leverage the information provided by $\Phi_n$, as it does not induce additional linearization due to evaluating $\Phi_n$ at points different from $h_n$.

Exactly solving the fixed-point problem is difficult. In the experiments, we adopt a heuristic which computes an approximation to $h_n^*$ as follows. We suppose $\Phi_n = \nabla f_n$ for some function $f_n$, which is the case e.g.,when $\Phi_n$ is the simulated gradient based on some (biased) dynamics model. This restriction makes the fixed-point problem as finding a stationary point of the optimization problem $\min_{\pi \in \Pi} f_n(\pi) + B_{R_{n-1}}(\pi || \hat{\pi}_n)$. In implementation, we initialize the iterate in this subproblem as $\texttt{update}(\hat{h}_n, H_{n-1}, \Phi_n(\hat{\pi}_n), w_n)$, which

is the output of the Prediction Step if we were to use $\hat{g}_n = \Phi_n(\hat{\pi}_n)$. We made this choice in initializing the subproblem, as we know that using $\hat{g}_n = \Phi_n(\hat{\pi}_n)$ in PICCOLO already works well (see the experiments) and it can be viewed as the solution to the fixed-point problem with respect to the linearized version of $\Phi_n$ at $\hat{\pi}_n$. Given the this initialization point, we proceed to compute the approximate solution to the fixed-point by applying the given base algorithm for $5$ iterations and then return the last iterate as the approximate solution. For example, if the base algorithm is natural gradient descent, we fixed the Bregman divergence (i.e.,its the Fisher information matrix as $\hat{\pi}_n$) and only updated the scalar stepsize adaptively along with the policy in solving this *regularized model-based RL problem* (i.e.,$\min_{\pi \in \Pi} f_n(\pi) + B_{R_{n-1}}(\pi || \hat{\pi}_n)$). While such simple implementation is not ideal, we found it works in practice, though we acknowledge that a better implementation of the subproblem solver would improve the results.

### B.8.2 Tasks

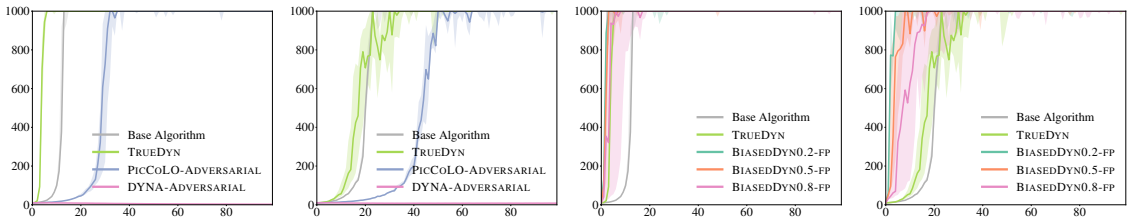
The robotic control tasks that are considered in the experiments are CartPole, Hopper, Snake, and Walker3D from OpenAI Gym [97] with the DART physics engine [98][§]. CartPole is a classic control problem, and its goal is to keep a pole balanced in a upright posture, by only applying force to the cart. Hopper, Snake, and Walker3D are locomotion tasks, of which the goal is to control an agent to move forward as quickly as possible without falling down (for Hopper and Walker3D) or deviating too much from moving forward (for Snake). Hopper is monopedal and Walker3D is bipedal, and both of them are subjected to significant contact discontinuities that are hard or even impossible to predict.

### B.8.3 Full Experimental Results

In Figure B.1, we empirically study the properties of PICCOLO that are predicted by theory on CartPole environment. In Figure B.2, we "PICCOLO " three base algorithms: ADAM,

[§]The environments are defined in DartEnv, hosted at https://github.com/DartEnv.

NATGRAD, TRPO, and apply them on four simulated environments: CartPole, Hopper, Snake, and Walker3D.




(a) Adv., NATGRAD (b) Adv. , TRPO (c) Fidelity, NATGRAD (d) Fidelity, TRPO

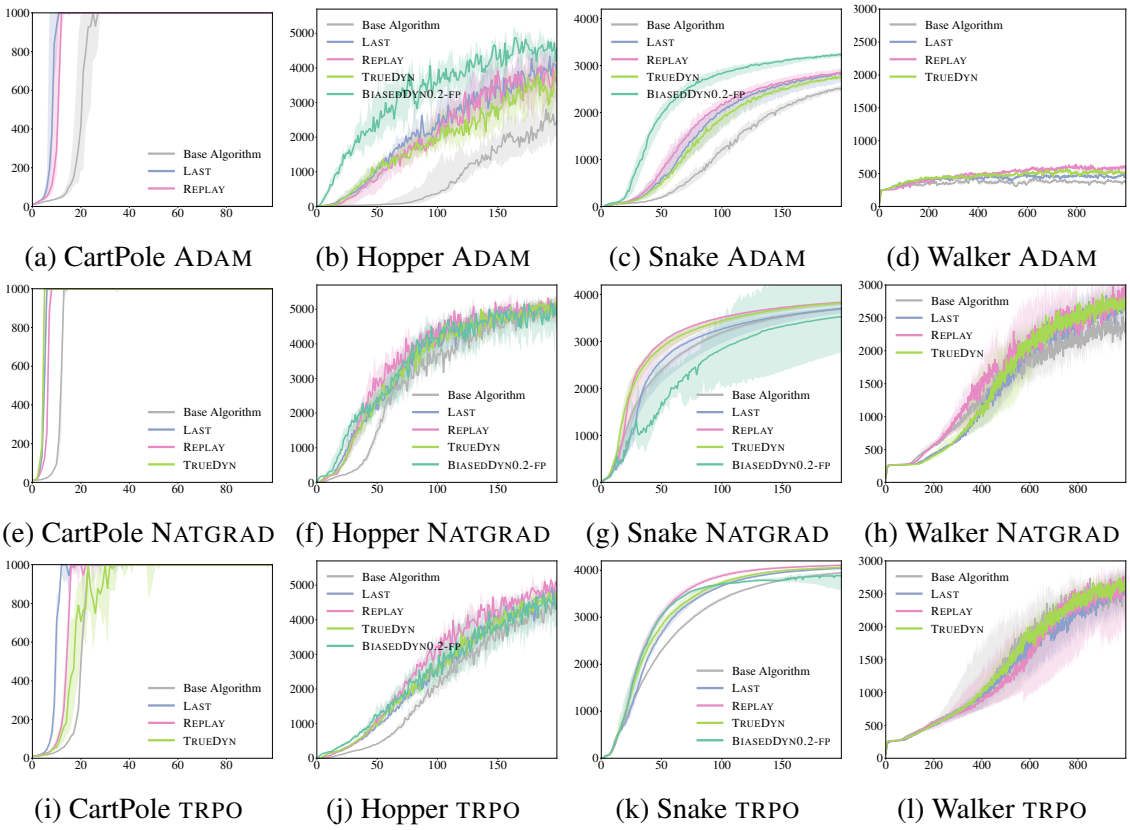
Figure B.1: The update rule, by default, is PICCOLO. For example TRUEDYN in (a) refers to PICCOLO with TRUEDYN predictive model. (a), (b): Comparison of PICCOLO and DYNA with adversarial model using NATGRAD and TRPO as base algorithms. (c), (d): PICCOLO with the fixed-point setting (4.8) with dynamics model in different fidelities. BIASEDDYN0.8 indicates that the mass of each individual robot link is either increased or decreased by $80\%$ with probability 0.5 respectively.

### B.8.4 Experiment Hyperparameters

The hyperparameters used in the experiments and the basic attributes of the environments are detailed in Table B.1.

Table B.1: Tasks specifics and hyperparameters.

| | CartPole | Hopper | Snake | Walker3D |
|---|---|---|---|---|
| Observation space dimension | 4 | 11 | 17 | 41 |
| Action space dimension | 1 | 3 | 6 | 15 |
| State space dimension | 4 | 12 | 18 | 42 |
| Number of samples from env. per iteration | 4k | 16k | 16k | 32k |
| Number of samples from model dyn. per iteration | 4k | 16k | 16k | 32k |
| Length of horizon | 1,000 | 1,000 | 1,000 | 1,000 |
| Number of iterations | 100 | 200 | 200 | 1,000 |
| Number of iterations of samples for REPLAY buffer | 5 | 4 | 3 | 2 (3 for ADAM) |
| $\alpha$ ¶ | 0.1 | 0.1 | 0.1 | 0.01 |
| $\eta$ in ADAM | 0.005 | 0.005 | 0.002 | 0.01 |
| $\eta$ in NATGRAD | 0.05 | 0.05 | 0.2 | 0.2 |
| $\eta$ in TRPO | 0.002 | 0.002 | 0.01 | 0.04 |




(a) CartPole ADAM (b) Hopper ADAM (c) Snake ADAM (d) Walker ADAM

(e) CartPole NATGRAD (f) Hopper NATGRAD (g) Snake NATGRAD (h) Walker NATGRAD

(i) CartPole TRPO (j) Hopper TRPO (k) Snake TRPO (l) Walker TRPO

Figure B.2: The performance of PICCOLO with different predictive models on various tasks, compared to base algorithms. The rows use ADAM, NATGRAD, and TRPO as the base algorithms, respectively. $x$ axis is iteration number and $y$ axis is sum of rewards. The curves are the median among 8 runs with different seeds, and the shaded regions account for $25\%$ percentile.

[¶] $\alpha$ and $\eta$ appear in the decaying step size multiplier for all the algorithms in the form $\eta/(1+\alpha\sqrt{n})$. $\alpha$ influences how fast the step size decays. We chose $\alpha$ in the experiments based on the number of iterations.

# APPENDIX C

# APPENDIX FOR CHAPTER 5

## C.1 Proof of Tool Lemmas

### C.1.1 Proof of Lemma 1

For completeness, we provide the proof for the basic inequality that upper bounds the norm of gradients by the function values, for smooth and nonnegative functions. This is essential for obtaining the self-bounding properties for proving Lemma 9 and Theorem 4 later on.

**Lemma 4** (Lemma 3.1 [111])**.** *Suppose a function $f : \mathcal{H} \to \mathbb{R}$ is $\beta$-smooth and non-negative, then for any $x \in \mathcal{H}$, $\|\nabla f(x)\|_*^2 \leq 4\beta f(x)$.*

*Proof.* Fix any $x \in \mathcal{H}$. And fix any $y \in \mathcal{H}$ satisfying $\|y - x\| \leq 1$. Let $g(u) = f(x + u(y - x))$ for any $u \in \mathbb{R}$. Fix any $u, v \in \mathbb{R}$,

$$\begin{aligned}
|g'(v) - g'(u)| &= |\langle \nabla f(x + v(y - x)) - \nabla f(x + u(y - x)), y - x\rangle| \\
&\leq \|\nabla f(x + v(y - x)) - \nabla f(x + u(y - x))\|_* \|y - x\| \\
&\leq \beta |v - u| \|y - x\|^2 \\
&\leq \beta |v - u|
\end{aligned}$$

Hence, $g$ is $\beta$-smooth. By the mean-value theorem, for any $u, v \in \mathbb{R}$, there exists $w \in (u, v)$, such that $g(v) = g(u) + g'(w)(v - u)$. Hence

$$\begin{aligned}
0 \leq g(v) &= g(u) + g'(u)(v - u) + (g'(w) - g'(u))(v - u) \\
&\leq g(u) + g'(u)(v - u) + \beta |w - u||v - u| \leq g(u) + g'(u)(v - u) + \beta (v - u)^2
\end{aligned}$$

Setting $v = u - \frac{g'(u)}{2\beta}$ yields that $|g'(u)| \leq \sqrt{4\beta g(u)}$. Therefore, we have

$$|g'(0)| = |\langle \nabla f(x), y - x \rangle| \leq \sqrt{4\beta g(0)} = \sqrt{4\beta f(x)}$$

Therefore, by the definition of dual-norm,

$$\|\nabla f(x)\|_* = \sup_{y \in \mathcal{B}, \|y-x\| \leq 1} \langle \nabla f(x), y - x \rangle = \sup_{y \in \mathcal{B}, \|y-x\| \leq 1} |\langle \nabla f(x), y - x \rangle| \leq \sqrt{4\beta f(x)}$$

□

It's worthy to note that $f$ needs to be smooth and non-negative on the entire Hilbert space $\mathcal{H}$.

### C.2 Proof of Theorem 3

**Theorem 3.** *In Algorithm 4, suppose $\{\hat{l}_n\}$ is CSN and the online algorithm $\mathcal{A}$ is admissible. Let $\hat{\epsilon} = \frac{1}{N} \min_{\theta \in \Theta} \sum \hat{l}_n(\theta)$ be the bias, and let $\hat{E}$ be such that $\hat{E} \geq \hat{\epsilon}$ almost surely. Choose $\eta$ for $\mathcal{A}$ to be $\frac{1}{2\left(\beta + \sqrt{\beta^2 + \frac{\beta N \hat{E}}{2R_{\mathcal{A}}^2}}\right)}$. Then in expectation*

$$\frac{1}{N} \sum l_n(\theta_n) - \hat{\epsilon} \leq \frac{8\beta R_{\mathcal{A}}^2}{N} + \sqrt{\frac{8\beta R_{\mathcal{A}}^2 \hat{E}}{N}} \tag{5.8}$$

The rate (5.8) follows from analyzing the regret and the generalization error in the decomposition in (5.6). First, under the assumption of CSN loss functions and admissible online algorithms, the online regret can be bounded by an extension of the bias-dependent regret bound that is stated for mirror descent in [111, Theorem 2], whose average gives the rate in (5.8) (see Appendix C.2.1). Second, the generalization error in (5.6) vanishes in expectation because it is a martingale difference sequence (see Appendix C.2.2).

### C.2.1 Upper Bound of Online Regret

We show a bias-dependent regret bound for admissible online algorithms (Definition 3) with CSN functions (Definition 4) by extending Theorem 2 of [111] as follows.

**Lemma 9.** *Consider running an admissible online algorithm $\mathcal{A}$ on a sequence of CSN loss functions $\{f_n\}$. Let $\{\theta_n\}$ denote the online decisions made in each round, and let $\hat{\epsilon} = \frac{1}{N}\min_{\theta\in\Theta}\sum f_n(\theta)$ be the bias, and let $\hat{E}$ be such that $\hat{E} \geq \hat{\epsilon}$ almost surely. Choose $\eta$ for $\mathcal{A}$ to be $\frac{1}{2\left(\beta+\sqrt{\beta^2+\frac{\beta N\hat{E}}{2R_{\mathcal{A}}^2}}\right)}$. Then the following holds*

$$\mathrm{Regret}(f_n) \leq 8\beta R_{\mathcal{A}}^2 + \sqrt{8\beta R_{\mathcal{A}}^2 N\hat{E}}.$$

*Proof.* Because the online algorithm $\mathcal{A}$ is admissible, we have

$$\mathrm{Regret}(f_n) \leq \frac{1}{\eta}R_{\mathcal{A}}^2 + \frac{\eta}{2}\sum\|\nabla l_n(\pi_n)\|_*^2 \tag{C.1}$$

Let $\lambda = \frac{1}{2\eta}$ and $r^2 = 2R_{\mathcal{A}}^2$, then

$$\frac{1}{\eta}R_{\mathcal{A}}^2 + \frac{\eta}{2}\sum\|\nabla l_n(\pi_n)\|_*^2 = \lambda r^2 + \sum\frac{1}{4\lambda}\|\nabla l_n(\pi_n)\|_*^2 \tag{C.2}$$

Using Lemma 4 yields a *self-bounding property* for $\mathrm{Regret}(f_n)$:

$$\mathrm{Regret}(f_n) \leq \lambda r^2 + \frac{\beta}{\lambda}\sum f_n(\theta_n) \leq \lambda r^2 + \frac{\beta}{\lambda}\mathrm{Regret}(f_n) + \frac{\beta}{\lambda}N\hat{E} \tag{C.3}$$

By rearranging the terms, we have a bias-dependent upper bound

$$\mathrm{Regret}(f_n) \leq \frac{\beta}{\lambda-\beta}N\hat{E} + \frac{\lambda^2}{\lambda-\beta}r^2 \tag{C.4}$$

The upper bound can be minimized by choosing an optimal $\lambda$. Setting the derivative of the

right-hand side to zero, and computing the optimal $\lambda$ ($\lambda > 0$) gives us

$$r^2\lambda^2 - 2\beta r^2\lambda - \beta N\hat{E} = 0, \quad \lambda > 0 \quad \text{and} \quad \lambda = \beta + \sqrt{\beta^2 + \frac{\beta N\hat{E}}{r^2}} \tag{C.5}$$

which implies that the optimal $\eta$ is $\frac{1}{2\left(\beta+\sqrt{\beta^2+\frac{\beta N\hat{E}}{2R_{\mathcal{A}}^2}}\right)}$. Since the optimal $\lambda$ satisfies $\beta N\hat{E} = r^2\lambda^2 - 2\beta r^2\lambda$ implied from (C.5), (C.4) can be simplified into:

$$\begin{aligned}\text{Regret}(f_n) &\leq \frac{1}{\lambda-\beta}\beta N\hat{E} + \frac{\lambda^2}{\lambda-\beta}r^2 = \frac{1}{\lambda-\beta}(r^2\lambda^2 - 2\beta r^2\lambda) + \frac{\lambda^2}{\lambda-\beta}r^2 \\ &= \frac{2\lambda^2 r^2 - 2\beta\lambda r^2}{\lambda-\beta} = 2\lambda r^2\end{aligned} \tag{C.6}$$

Plugging in the optimal $\lambda$ yields

$$\begin{aligned}\text{Regret}(f_n) &\leq 2\lambda r^2 = 2\left(\beta + \sqrt{\beta^2 + \frac{\beta N\hat{E}}{r^2}}\right) r^2 \\ &= 2\beta r^2 + \sqrt{2\beta r^2}\sqrt{2\beta r^2 + 2N\hat{E}} \\ &\leq 4\beta r^2 + 2\sqrt{\beta r^2 N\hat{E}} \\ &= 8\beta R_{\mathcal{A}}^2 + \sqrt{8\beta R_{\mathcal{A}}^2 N\hat{E}}\end{aligned} \tag{C.7}$$

where the last inequality uses the basic inequality: $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$. □

Notably, the admissibility defined in Definition 3 is satisfied by common online algorithms, such as mirror descent [117] and Follow-The-Regularized-Leader [96] under first-order or full-information feedback, where $\eta$ in Definition 3 corresponds to a constant stepsize, and $R_{\mathcal{A}}$ measures the size of the decision set $\Theta$. More concretely, assume that the loss functions $\{f_n\}$ are convex. Then for mirror descent, with constant stepsize $\eta$, i.e., $\theta_{n+1} = \arg\min_{\theta\in\Theta} f_n(\theta) + \frac{1}{\eta}D_h(\theta||\theta_n)$, where $h$ is 1-strongly convex and $D_h$ is the Bregman distance generated by $h$ defined by $D_h(x||y) = h(x) - h(y) - \langle\nabla h(y), x - y\rangle$ [180], $R_{\mathcal{A}}^2$ can be set to $\max_{x,y\in\Theta} D_h(x||y)$. And for FTRL with constant stepsize $\eta$, i.e., $\theta_{n+1} =$

$\arg\min_{\theta\in\Theta}\sum f_n(\theta)+\frac{1}{\eta}h(\theta)$, where $h$ is 1-strongly convex and non-negative, $R_{\mathcal{A}}^2$ can be set to $\max_{\theta\in\Theta}h(\theta)$ [96, Theorem 1].

### C.2.2 The Generalization Error Vanishes in Expectation

The generalization error in (5.6) vanishes in expectation because it is a martingale difference sequence.

**Lemma 10.** *For Algorithm 4, the following holds:* $\mathbb{E}[\sum l_n(\theta_n)-\sum\hat{l}_n(\theta_n)]=0$.

*Proof.* We show this by working from the end of the sequence.

$$\begin{aligned}
\mathbb{E}_{\hat{l}_{1:N}}\left[\sum_{t=1}^{N}l_n(\theta_n)\right] &= \mathbb{E}_{\hat{l}_{1:N-1}}\left[\sum_{t=1}^{N-1}l_n(\theta_n)+l_N(\theta_N)\right]\\
&= \mathbb{E}_{\hat{l}_{1:N-1}}\left[\sum_{t=1}^{N-1}l_n(\theta_n)+\mathbb{E}_{\hat{l}_N|\hat{l}_{1:N-1}}\left[\hat{l}_N(\theta_N)\right]\right]\\
&= \mathbb{E}_{\hat{l}_{1:N-2}}\left[\sum_{t=1}^{N-2}l_n(\theta_n)+l_{N-1}(\theta_{N-1})+\mathbb{E}_{\hat{l}_{N-1:N}|\hat{l}_{1:N-2}}\left[\hat{l}_N(\theta_N)\right]\right]\\
&= \mathbb{E}_{\hat{l}_{1:N-2}}\left[\sum_{t=1}^{N-2}l_n(\theta_n)+\mathbb{E}_{\hat{l}_{N-1}|\hat{l}_{1:N-2}}\left[\hat{l}_{N-1}(\theta_{N-1})\right]+\mathbb{E}_{\hat{l}_{N-1:N}|\hat{l}_{1:N-2}}\left[\hat{l}_N(\theta_N)\right]\right]\\
&= \mathbb{E}_{\hat{l}_{1:N-2}}\left[\sum_{t=1}^{N-2}l_n(\theta_n)+\mathbb{E}_{\hat{l}_{N-1:N}|\hat{l}_{1:N-2}}\left[\sum_{t=N-1}^{N}\hat{l}_n(\theta_n)\right]\right]
\end{aligned}$$

By applying the steps above repeatedly, the desired equality can be obtained. □

### C.2.3 Putting Together

Finally, plugging Lemma 9 and Lemma 10 into (5.6) yields (5.8).

## C.3 Proof of Theorem 4

**Theorem 4.** *Under the same assumptions and setup of Theorem 3, further assume that there is* $G\in[0,\infty)$ *such that, for any* $\theta\in\Theta$, $\|\nabla\hat{l}_n(\theta)\|_*\le G$. *For* $\delta<1/e$, *with probability at*

*least* $1-\delta$*,*

$$\frac{1}{N}\sum l_n(\theta_n) - \epsilon = O\left(\frac{C\beta R^2}{N} + \sqrt{\frac{C\beta R^2(\hat{E}+\epsilon)}{N}}\right) \tag{5.9}$$

*where* $R_\Theta = \max_{\theta\in\Theta}\|\theta\|, R = \max(1, R_\Theta, R_\mathcal{A}), C = \log(1/\delta)\log(GRN)$.

### C.3.1 Decomposition

The key to avoid the slow rate due to the direct application of martingale concentration analyses on the MDSs in (5.6) and (5.7) is to take a different decomposition of the cumulative loss. Here we construct two *new* MDSs in terms of the gradients: recall $\epsilon = \min_{\theta\in\Theta}\sum l_n(\theta)$ and let $\theta^\star = \arg\min_{\theta\in\Theta}\sum l_n(\theta)$. Then by convexity of $l_n$, we can derive

$$\begin{aligned}
&\sum l_n(\theta_n) - N\epsilon \\
\le& \sum \langle \nabla l_n(\theta_n), \theta_n - \theta^\star\rangle \\
=& \sum \langle \nabla l_n(\theta_n) - \nabla\hat{l}_n(\theta_n), \theta_n - \theta^\star\rangle + \sum\langle \nabla\hat{l}_n(\theta_n), \theta_n - \theta^\star\rangle \\
\le& \sum \underbrace{\langle \nabla l_n(\theta_n) - \nabla\hat{l}_n(\theta_n), \theta_n\rangle}_{\text{MDS}} - \sum\underbrace{\langle \nabla l_n(\theta_n) - \nabla\hat{l}_n(\theta_n)}_{\text{MDS}}, \theta^\star\rangle + \text{Regret}(\langle\nabla\hat{l}_n(\theta_n), \cdot\rangle)
\end{aligned} \tag{C.8}$$

Our proof is based on analyzing these three terms. The two MDSs are analyzed in Appendix C.3.2 and the regret is analyzed in Appendix C.3.3.

### C.3.2 Upper Bound of the Martingale Concentration

For the MDSs in (C.8), we notice that, for smooth and non-negative functions, the squared norm of the gradient can be bounded by the corresponding function value through Lemma 4. This enables us to properly control the second-order statistics of the MDSs in (C.8). By a recent vector-valued martingale concentration inequality that depends only on the second-order statistics [114], we obtain a self-bounding property for (C.8) to get fast concentration

rate. The martingale concentration inequality is stated in the following lemma.

**Lemma 11** (Theorem 3 [114]). *Let $\mathcal{K}$ be a Hilbert space with norm $\|\cdot\|$ whose dual is $\|\cdot\|_*$. Let $\{z_t\}$ be a $\mathcal{K}$-valued martingale difference sequence with respect to $\{y_t\}$, i.e., $\mathbb{E}_{z_t|y_1,\ldots,y_{t-1}}[z_t] = 0$, and let $h$ be a 1-strongly convex function with respect to norm $\|\cdot\|$ and let $B^2 = \sup_{x,y\in\mathcal{K},\|x\|=1,\|y\|=1} D_h(x||y)$. Then for $\delta \leq 1/e$, with probability at least $1-\delta$*

$$\left\|\sum z_t\right\|_* \leq 2B\sqrt{V} + \sqrt{2\log(1/\delta)}\sqrt{1+1/2\log(2V+2W+1)}\sqrt{2V+2W+1}$$

*where $V = \sum \|z_t\|_*^2$ and $W = \sum \mathbb{E}_{z_t|y_1,\ldots,y_{t-1}}\|z_t\|_*^2$.*

In order to apply Lemma 11 to the MDSs in (C.8), the key is to properly upper bound the statistics $V$ and $W$ in Lemma 11 for these MDSs.

*Upper Bound of the Concetration for MDS $\langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n), \theta_n\rangle$*

Suppose that the decision set $\Theta$ is inside a ball centered at the origin in $\mathcal{H}$ with radius $R_\Theta$.

**Assumption 5.** There exists $R_\Theta \in [0,\infty)$, such that $\max_{\theta\in\Theta}\|\theta\| \leq R_\Theta$.

Then by the definition of $V$ and $W$ in Lemma 11, and the definitions of the two problem-dependent policy class biases $\epsilon$ and $\hat{\epsilon}$ (see Definition 2), one can obtain

$$\begin{aligned}
V &= \sum |\langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n), \theta_n\rangle|^2 \\
&\leq \sum R_\Theta^2 \|\nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)\|_*^2 && \text{Assumption 5} \\
&\leq \sum R_\Theta^2 \left(2\|\nabla l_n(\theta_n)\|_*^2 + 2\|\nabla \hat{l}_n(\theta_n)\|_*^2\right) && \text{triangle inequality} \qquad \text{(C.9)} \\
&\leq \sum R_\Theta^2 \left(8\beta l_n(\theta_n) + 8\beta \hat{l}_n(\theta_n)\right) && \text{Lemma 4} \\
&= 8\beta R_\Theta^2 (\text{Regret}(l_n) + \text{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}) && \text{Definition 2} \qquad \text{(C.10)}
\end{aligned}$$

Similarly for $W$, we have

$$
\begin{aligned}
W &= \sum \mathbb{E}_{\hat{l}_n|\theta_n}[|\langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n), \theta_n \rangle|^2] \\
&\leq \sum R_\Theta^2 \mathbb{E}_{\hat{l}_n|\theta_n}[\|\nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)\|_*^2] && \text{Assumption 5} \\
&\leq \sum R_\Theta^2 \left(2\|\nabla l_n(\theta_n)\|_*^2 + 2\mathbb{E}_{\hat{l}_n|\theta_n}[\|\nabla \hat{l}_n(\theta_n)\|_*^2]\right) && \text{triangle inequality} \qquad \text{(C.11)} \\
&\leq \sum R_\Theta^2 \left(8\beta l_n(\theta_n) + 8\mathbb{E}_{\hat{l}_n|\theta_n}[\beta \hat{l}_n(\theta_n)]\right) && \text{Lemma 4} \\
&= \sum R_\Theta^2 \left(8\beta l_n(\theta_n) + 8\beta l_n(\theta_n)\right) \\
&= 16\beta R_\Theta^2 (\text{Regret}(l_n) + N\epsilon) && \text{Definition 2} \qquad \text{(C.12)}
\end{aligned}
$$

Therefore,

$$
V + W \leq 24\beta R_\Theta^2 (\text{Regret}(l_n) + \text{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}) \qquad \text{(C.13)}
$$

Further suppose that the gradient of the sampled loss can be uniformly bounded:

**Assumption 6.** For any loss sequence $\{\hat{l}_n\}$ that can be experienced by Algorithm 4, suppose that there is $G \in [0, \infty)$ such that, for any $\theta \in \Theta$, $\|\nabla \hat{l}_n(\theta)\|_* \leq G$.

Then due to (C.13), $V \leq 4G^2 R_\Theta^2 N$ and $W \leq 4G^2 R_\Theta^2 N$. Now we are ready to invoke Lemma 11 by letting the Hilbert space $\mathcal{K}$ in Lemma 11 be $\mathbb{R}$, and denoting the corresponding $B$ in Lemma 11 by $B_\mathbb{R}$. Then, for $\delta > 1/e$, with probability at least $1 - \delta$, the following holds

$$
\begin{aligned}
&|\sum \langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n), \theta_n \rangle| \\
&\leq 2B_\mathbb{R} \sqrt{8\beta R_\Theta^2} \sqrt{\text{Regret}(l_n) + \text{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}} + \\
&\sqrt{96\beta R_\Theta^2 \log(1/\delta)} \sqrt{1 + 1/2 \log(16G^2 R_\Theta^2 N + 1)} \cdot \qquad \text{(C.14)} \\
&\sqrt{\text{Regret}(l_n) + N\epsilon + \text{Regret}(\hat{l}_n) + N\hat{\epsilon} + 1/(48\beta R_\Theta^2)} \qquad \text{(C.15)}
\end{aligned}
$$

*Upper Bound of the Concetration for MDS* $\nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)$

To bound $\| \sum \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)\|_*$ that appears in

$$\sum \langle \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n), \theta^\star \rangle \leq R_\Theta \| \sum \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)\|_* \tag{C.16}$$

We use Lemma 11 again in a similar way of deriving (C.14), except that this time the MDS $\nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)$ is vector-valued. Akin to showing (C.10) and (C.12), the statistics $V$ can be bounded as

$$V \leq \sum \left( 2\|\nabla l_n(\theta_n)\|_*^2 + 2\|\nabla \hat{l}_n(\theta_n)\|_*^2 \right) \tag{C.17}$$

$$\leq 8\beta(\text{Regret}(l_n) + \text{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}) \tag{C.18}$$

and similarly for $W$:

$$W \leq \sum \left( 2\|\nabla l_n(\theta_n)\|_*^2 + 2\mathbb{E}_{\hat{l}_n|\theta_n}[\|\nabla \hat{l}_n(\theta_n)\|_*^2] \right) \tag{C.19}$$

$$\leq 16\beta(\text{Regret}(l_n) + N\epsilon) \tag{C.20}$$

Therefore,

$$V + W \leq 24\beta(\text{Regret}(l_n) + \text{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}) \tag{C.21}$$

Furthermore, by Assumption 6, it can be shown from (C.17) and (C.19) that $V \leq 4G^2N$ and $W \leq 4G^2N$. To invoke Lemma 11, let $\mathcal{K}$ in Lemma 11 be $\mathcal{H}$, and denote the corresponding $B$ in Lemma 11 by $B_\mathcal{H}$. Then, for $\delta < 1/e$, with probability at least $1 - \delta$, the following

holds

$$\begin{aligned}
&\|\sum \nabla l_n(\theta_n) - \nabla \hat{l}_n(\theta_n)\|_* \\
&\leq 2B_{\mathcal{H}}\sqrt{8\beta}\sqrt{\mathrm{Regret}(l_n) + \mathrm{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}} + \\
&\quad \sqrt{96\beta\log(1/\delta)}\sqrt{1 + 1/2\log(16G^2N+1)}\sqrt{\mathrm{Regret}(l_n) + N\epsilon + \mathrm{Regret}(\hat{l}_n) + N\hat{\epsilon} + 1/(48\beta)}
\end{aligned} \tag{C.22}$$

### C.3.3 Upper Bound of the Regret

Besides analyzing the MDSs, we need to bound the regret to the linear functions defined by the gradients (the last term in (C.8)). Since this last term is linear, not CSN, the bias-dependent online regret bound in the proof of Theorem 3 does not apply. Nonetheless, because these linear functions are based on the gradients of CSN functions, we discover that their regret rate actually obeys the exact same rate as the regret to the CSN loss functions. This is notable because the regret to these linear functions upper bounds the regret to the CSN loss functions.

**Lemma 12.** *Under the same assumptions and setup in Lemma 9,*

$$\mathrm{Regret}(\langle \nabla l_n(\pi_n), \cdot \rangle) \leq 8\beta R_{\mathcal{A}}^2 + \sqrt{8\beta R_{\mathcal{A}}^2 N\hat{E}}. \tag{C.23}$$

*Proof.* It suffices to show a self-bounding property for $\mathrm{Regret}(\langle \nabla l_n(\pi_n), \cdot \rangle)$ as (C.3). Once this is established, the rest resembles how (C.7) follows from (C.3) through algebraic manipulations. As in Lemma 9, define $\lambda = \frac{1}{2\eta}$ and $r^2 = 2R_{\mathcal{A}}^2$. Due to the property of admissible online algorithms, one can obtain

$$\mathrm{Regret}(\langle \nabla l_n(\pi_n), \cdot \rangle) \leq \frac{1}{\eta}R_{\mathcal{A}}^2 + \frac{\eta}{2}\sum \|\nabla l_n(\pi_n)\|_*^2 = \lambda r^2 + \sum \frac{1}{4\lambda}\|\nabla l_n(\pi_n)\|_*^2 \tag{C.24}$$

To proceed, as in Lemma 9, let $\hat{\epsilon} = \frac{1}{N}\min_{\theta\in\Theta}\sum \hat{l}_n(\theta)$ be the bias, and let $\hat{E}$ be such that

$\hat{E} \geq \hat{\epsilon}$ almost surely. Using Lemma 4 and the admissibility of online algorithm $\mathcal{A}$ yields a self-bounding property for $\mathrm{Regret}(\langle \nabla l_n(\pi_n), \cdot \rangle)$:

$$\begin{aligned}
\mathrm{Regret}(\langle \nabla l_n(\pi_n), \cdot \rangle) &\leq \lambda r^2 + \frac{\beta}{\lambda} \sum f_n(\theta_n) \\
&\leq \lambda r^2 + \frac{\beta}{\lambda} \mathrm{Regret}(f_n) + \frac{\beta}{\lambda} N \hat{E} \\
&\leq \lambda r^2 + \frac{\beta}{\lambda} \mathrm{Regret}(\langle \nabla l_n(\pi_n), \cdot \rangle) + \frac{\beta}{\lambda} N \hat{E}
\end{aligned}$$

This self-bounding property is exactly like what we have seen in the self-bounding property for $\mathrm{Regret}(f_n)$. After rearranging and computing the optimal $\lambda$ (which coincides with the optimal $\lambda$ in Lemma 9), (C.23) follows. □

Lemma 12 provides a bias-dependent regret to the linear functions defined by the gradients when the (stepsize) constant $\eta$ is set optimally in the online algorithm $\mathcal{A}$ (used in Algorithm 4). Interestingly, the optimal $\eta$ that achieves the bias-dependent regret coincides with the one for achieving a bias-dependent regret to CSN functions. Therefore, a bias-dependent bound for $\mathrm{Regret}(\hat{l}_n)$ and $\mathrm{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle)$ can be achieved simultaneously.

### C.3.4 Putting Things Together

We now have all the pieces to prove Theorem 4. Plugging (C.14), (C.16), and (C.22) into the decomposition (C.8), we have, for $\delta < 1/e$, with probability at least $1 - 2\delta$

$$
\begin{aligned}
&\mathrm{Regret}(l_n) \\
&\leq 2B_{\mathbb{R}}\sqrt{8\beta R_\Theta^2}\sqrt{\mathrm{Regret}(l_n) + \mathrm{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}} \\
&+ \sqrt{96\beta R_\Theta^2 \log(1/\delta)}\sqrt{1 + 1/2\log(16G^2R_\Theta^2 N + 1)}\cdot \\
&\quad \sqrt{\mathrm{Regret}(l_n) + N\epsilon + \mathrm{Regret}(\hat{l}_n) + N\hat{\epsilon} + 1/(48\beta R_\Theta^2)} \\
&+ 2B_{\mathcal{H}}\sqrt{8\beta R_\Theta^2}\sqrt{\mathrm{Regret}(l_n) + \mathrm{Regret}(\hat{l}_n) + N\epsilon + N\hat{\epsilon}} \\
&+ \sqrt{96\beta \log(1/\delta)}\sqrt{1 + 1/2\log(16G^2N + 1)}\sqrt{\mathrm{Regret}(l_n) + N\epsilon + \mathrm{Regret}(\hat{l}_n) + N\hat{\epsilon} + 1/(48\beta)} \\
&+ \mathrm{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle)
\end{aligned}
$$

To simplify it, we denote

$$
\begin{aligned}
A_1 &= 8\max(B_{\mathbb{R}}, B_{\mathcal{H}})\sqrt{2\beta R_\Theta^2}, \\
A_2 &= 8\sqrt{6\beta R_\Theta^2 \log(1/\delta)}\sqrt{1 + 1/2\log(16G^2\max(1, R_\Theta^2)N + 1)}, \\
\tilde{R} &= \min(1, R_\Theta)
\end{aligned}
$$

Plugging them into the above upper bound on $\mathrm{Regret}(l_n)$ and using the basic inequality $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ yield

$$
\begin{aligned}
\mathrm{Regret}(l_n) \leq (A_1 + A_2)\sqrt{\mathrm{Regret}(l_n) + \mathrm{Regret}(\hat{l}_n)} + (A_1 + A_2)\sqrt{N\epsilon + N\hat{\epsilon}} \\
+ \frac{A_2}{\sqrt{48\beta\tilde{R}^2}} + \mathrm{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle)
\end{aligned}
$$

To further simplify, using the basic inequality $\sqrt{ab} \leq (a+b)/2$ yields

$$\begin{aligned}\mathrm{Regret}(l_n) \leq \frac{\mathrm{Regret}(l_n)}{2} + \frac{\mathrm{Regret}(\hat{l}_n)}{2} + (A_1 + A_2)\sqrt{N\epsilon + N\hat{\epsilon}} \\ + \frac{(A_1 + A_2)^2}{2} + \frac{A_2}{\sqrt{48\beta\tilde{R}^2}} + \mathrm{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle)\end{aligned}$$

Rearranging terms and invoking the bias-dependent rate in Lemma 9 and Lemma 12 give

$$\begin{aligned}\mathrm{Regret}(l_n) &\leq \mathrm{Regret}(\hat{l}_n) + 2(A_1 + A_2)\sqrt{N\epsilon + N\hat{\epsilon}} + (A_1 + A_2)^2 + \frac{A_2}{\sqrt{12\beta\tilde{R}^2}} \\ &\quad + 2\mathrm{Regret}(\langle \nabla \hat{l}_n(\theta_n), \cdot \rangle) \\ &\leq 2(A_1 + A_2)\sqrt{N\epsilon + N\hat{\epsilon}} + 6\sqrt{2\beta R_{\mathcal{A}}^2 N\hat{E}} + (A_1 + A_2)^2 + \frac{A_2}{\sqrt{12\beta\tilde{R}^2}} + 24\beta R_{\mathcal{A}}^2\end{aligned}$$

Finally, to derive a big-$O$ bound, denote

$$R = \max(1, R_\Theta, R_{\mathcal{A}}), \quad C = \log(1/\delta)\log(GRN)$$

then one can obtain the rate *in terms of* $N$ in big-$O$ notation, while keeping $\tilde{R}$, $R$, $B_{\mathbb{R}}$, $B_{\mathcal{H}}$, $\log(1/\delta)$, $G$, $\epsilon$, and $\hat{E}$ as multipliers:

$$\mathrm{Regret}(l_n) = O\left(C\beta R^2 + \sqrt{C\beta R^2 N(\hat{E} + \epsilon)}\right)$$

Therefore

$$\frac{1}{N}\sum l_n(\theta_n) - \epsilon = O\left(\frac{C\beta R^2}{N} + \sqrt{\frac{C\beta R^2(\hat{E} + \epsilon)}{N}}\right)$$

## C.4 Experiment Details

Although the main focus of this work is the new theoretical insights, we conduct experiments to provide evidence that the fast policy improvement phenomena indeed exist, as our theory predicts. We verify the change of rates due to policy class capacity by running an online IL experiment in the CartPole balancing task in OpenAI Gym [97] with DART physics engine [98].

### C.4.1 MDP Setup

The goal of the CartPole balancing task is to keep the pole upright by controlling the acceleration of the cart. This MDP has a 4-dimensional continuous state space (the position and the velocity of the cart and the pole), and 1-dimensional continuous action space (the acceleration of the cart). The initial state is a configuration with a small uniformly sampled offset from being static and vertical, and the dynamics is deterministic. This task has a maximum horizon of $1000$. In each time step, if the pole is maintained within a threshold from being upright, the learner receives an instantaneous reward of one; otherwise, the learner receives zero reward and the episode terminates. Therefore, the maximum sum of rewards for an episode is $1000$.

### C.4.2 Expert Policy Representation and Training

To simulate the online IL task, we consider a neural network expert policy (with one hidden layer of $64$ units and $\tanh$ activation), and the inputs to the neural network is normalized using a moving average over the samples. The expert policy is trained using a model-free policy gradient method (ADAM [95] with GAE [101]). And the value function used by GAE is represented by a neural network with two hidden layers of $128$ units and $\tanh$ activation. To compute the policy gradient during training, additional Gaussian noise (with zero mean and a learnable variance that does not depend on the state) is added to the actions, and the

gradient is computed through log likelihood ratio. After $100$ rounds of training, the expert policy can consistently achieve the maximum sum of rewards both with and without the additional Gaussian noise. After the expert policy is trained, during online IL, Gaussian noise is not added in order to reduce the variance in the experiments.

### C.4.3 Learner Policy Representation

We let the learner policy be another neural network that has exactly the same architecture as the expert policy with no Gaussian noise added; we copy the weights for the hidden layer and the input normalizer from those of the expert policy and randomly initialized the weights of the output layer. During training, only the weights of the learner's output layer were updated. In this way, we can view the learner as a *linear* policy using the representation of the expert policy.

### C.4.4 Online IL Setup

**Policy class** We conduct online IL with unbiased and biased policy classes. One one hand, we define the unbiased class as all the policies satisfying the representation in Section C.4.3. On the other hand, we define the biased policy class by imposing an additional $\ell_2$-norm constraint on the learner's weights in the second layer so that the learner cannot perfectly mimic the expert policy. More concretely, in the experiments, the $\ell_2$-norm constraint is set to $0.1$, whereas the $\ell_2$-norm of the final policy trained without the constraint is $0.56$.

**Loss functions** We select $l_n(\theta) = \mathbb{E}_{s\sim d_{\pi_{\theta_n}}}[H_\mu(\pi_\theta(s) - \pi_{\mathrm{e}}(s))]$ as the online IL loss (see Section 5.2.2), where $H_\mu$ is the Huber function defined as $H_\mu(x) = \frac{1}{2}x^2$ for $|x| \leq \mu$ and $\mu|x| - \frac{1}{2}\mu^2$ for $|x| > \mu$. In the experiments, $\mu$ is set to $0.05$; as a result, $H_\mu$ is linear when its function value is larger than $0.00125$. Because the learner's policy is linear, this online loss is CSN in the unknown weights of the learner.

**Policy update rule** We choose ADAM [95], which is a first-order mirror descent algorithm, as the online algorithm in Algorithm 4. When the $\ell_2$-norm constraint is imposed, an additional projection step is taken after taking a gradient step using ADAM. The final algorithm is a special case of the DAgger algorithm [78] (called DAggereD in [86]) with only first-order information and continuous actions [86]. In the experiments, the stepsize is set to $0.01$. In each round, for updating the learner policy, $1000$ samples, i.e., state and expert action pairs, are gathered, and for computing the loss $l_n(\theta_n)$, more samples ($5000$ samples) are used due to the randomness in the initial state of the MDP.

**Hyperparameter tuning** The hyperparameters are tuned in a very coarse manner. We eliminated the ones that are obviously not proper. Here are the hyperparameters we have experimented. The stepsize in online IL: $0.1, 0.01, 0.001$. The $\ell_2$-norm constraint for biased policies: $0.1, 0.4$. The Huber function parameter $\mu$: $0.05$.

### C.4.5 Curve Fitting and Simulation results

We compare the learning results in the unbiased and biased settings, in terms of how the average loss $\frac{1}{N}\sum_{n=1}^{N} l_n(\theta_n)$ changes as the number of rounds $N$ in online learning increases. Due to the randomness in the initial state of the MDP, we used the median of the average loss from 4 random seeds. To see whether the rates of the learning curves are $O(1/N)$ or $O(1/\sqrt{N})$, we fit the curves of the average loss using the parametric functions $f_1(N) = a\frac{1}{N} + b$ and $f_2(N) = a\frac{1}{\sqrt{N}} + b$.

The parameters $a, b$ in the parametric functions are obtained by solving a constrained convex program using the python package CVXPY [181, 182]. We chose the loss function of the convex program to be the $\ell_1$-loss, in order to avoid over-penalizing the error at first several rounds and to capture the overall rate of the curves. The constraint of the convex program is that the graphs of the parametric functions must lie above the learning curves. The constraint is imposed because the rate predicted by the new theory is an upper bound.

The experimental results were depicted in Figure 5.1. In the unbiased setting (Figure 5.1a), the curve fits well with the parametric function $f_1$ in $O(1/N)$. By contrast, in the biased setting (Figure 5.1b), the curve aligns better with the parametric function $f_2$ in $O(1/\sqrt{N})$.

### C.4.6 Other Details

**Computing infrastructure** All the experiments were conducted on a desktop with Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 32GB memory, and no GPU. The operating system is Ubuntu 16.04.

**Average runtime** On the aforementioned desktop, it took 15 min to train the expert, 45 min to do online IL, and 3 sec to fit and plot the curves.

# APPENDIX D

# APPENDIX FOR CHAPTER 6

## D.1 Missing Proofs

### D.1.1 Proof for Theorem 6

To understand how the ordering matters, we consider a toy example of estimating $\mathbb{E}_{x,y}[f(x,y)]$ of some function $f$ of two random variables $x$ and $y$. We prove a basic lemma.

**Lemma 13.** *If $x$ and $y$ are independent, then*

$$\mathbb{V}_x\mathbb{E}_y\left[f(x,y)\right] \leq \mathbb{E}_y\mathbb{V}_x\left[f(x,y)\right] \tag{D.1}$$

*Proof.* This can be proved by Jensen's inequality.

$$\mathbb{V}_x\mathbb{E}_y\left[f\right] = \mathbb{E}_x\left(\mathbb{E}_y\left[f - \mathbb{E}_x\left[f\right]\right]\right)^2 \leq \mathbb{E}_x\mathbb{E}_y\left[\left(f - \mathbb{E}_{x,y}\left[f\right]\right)^2\right] = \mathbb{E}_y\mathbb{V}_x\left[f(x,y)\right]$$

□

Suppose we want to reduce the variance of estimating $\mathbb{E}_{x,y}[f(x,y)]$ with some CV $\phi(x,y)$ but only knowing the distribution $P(x)$, not $P(y)$. Lemma 13 tells us that in decomposing the total variance of $f(x,y)$ to design this CV (cf. Section 6.2.3) we should take the decomposition

$$\mathbb{V}_y\mathbb{E}_x\left[f(x,y)\right] + \mathbb{E}_y\mathbb{V}_x\left[f(x,y)\right] \tag{D.2}$$

instead of the decomposition

$$\mathbb{V}_x\mathbb{E}_y\left[f(x,y)\right]+\mathbb{E}_x\mathbb{V}_y\left[f(x,y)\right] \tag{D.3}$$

In other words, we should take the ordering $y \to x$, instead of $x \to y$, when invoking the law of total variance. The reason is that after choosing the optimal CV for each case to reduce the variance due to $x$ (the information that we have access to), we are left with $\mathbb{V}_y\mathbb{E}_x\left[f(x,y)\right]$ and $\mathbb{E}_x\mathbb{V}_y\left[f(x,y)\right]$, respectively, for $y \to x$ and $x \to y$. By Lemma 13, we see the $y \to x$ has a smaller residue in variance. In other words, when we only have partial information about the distribution, we should arrange the random variables whose distribution we know to the latter stage of the ordering, so that the CV we design can leverage the sampled observations to compensate for the lack of prior.

We use this idea to prove the natural ordering (6.10) in optimal. In analogy of $x$ and $y$, we have the action randomness whose distribution is known (i.e.the policy) and the dynamics randomness, whose distribution is unknown.

The potential orderings we consider come from first reparameterizing the policy and then ordering the independent random variables $\xi_t$ (cf. Section 6.4.4). The Bayes networks of the MDP with and without policy reparameterization are depicted in Figure D.1, based on which we draw conditional independent relations later in the proof. We note that the CV is determined by the ordering, not due to reparameterization. For the natural ordering,

$$s_t \to a_t \to s_{t+1} \to a_{t+1} \to \cdots \to s_T \to a_T, \tag{6.10}$$

it gives the same control variate of the ordering below based on reparameterization

$$s_t \to \xi_t \to s_{t+1} \to \xi_{t+1} \to \cdots \to s_T \to \xi_T. \tag{D.4}$$

Suppose that given an ordering, we can compute its optimal CV. We define the variance

left after applying that optimal CV associated with the ordering, the *residue* of that ordering. We will show that the residue is minimized at the natural ordering.

The proof consists of two steps.

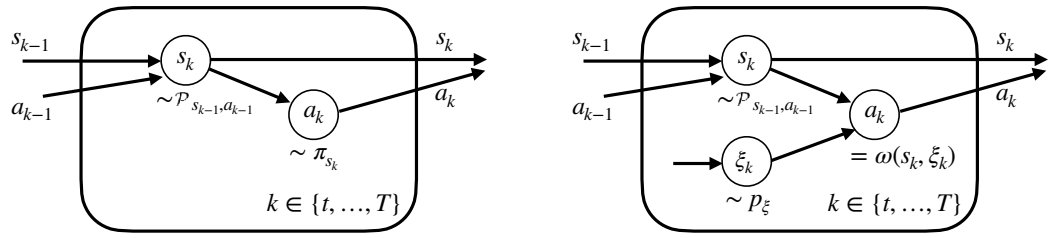
1. We show that when dynamics is the MDP is unknown, an ordering is *feasible* to implement, if and only if, $\xi_k$ appears before $s_{k+1..T}$ for all $t \leq k < T$. That is, a feasible ordering must be causal at least in actions: the action randomness that causes a state must be arranged before that state in the ordering. We prove this by contradiction. Assume otherwise $s_u$ is the *first* state before $\xi_k$ satisfying $u > k$. We see that $\xi_k$ and $s_u$ are dependent, if none of the variables in $s_{k+1..u}$ is given. This observation can been inferred from the Bayes network that connect these random variables (Figure D.1b), i.e.the path from $\xi_t$ to $s_u$ is not blocked unless any of $s_{k+1..u}$ is observed [183]. Therefore, if we have an ordering that is violates the causality property defined above, the expectation over $\xi_k$ required to define the difference estimator becomes intractable to compute, because the dynamics is *unknown*. This creates a contradiction.

2. We show that any feasible ordering can be transformed into the natural ordering in (6.10) using operations that do not increase the variance residue. We consider the following two operations

   (a) Suppose, in an ordering, there is $s_v \to s_u, v > u$, then we can exchange them without affecting variance residue.

   (b) Suppose, in a feasible ordering, there is $s_v \to \xi_k \to s_u$ with $v > u$ and $k \neq u, v$. Because this is a feasible ordering, we have $k + 1 \leq u < v$. This means that we can also move $\xi_k$ after $s_u$. This change would not increase variance residue, because of the discussion after Lemma 13. Then we change exchange the order of $s_v$ and $s_u$ too using the first operation.

   By using these two operations repeatedly, we can make all the states ordered by their




(a) before policy reparameterization (b) after policy reparameterization

Figure D.1: Bayes networks for the random variables in $g_t$ (6.2), before and after reparameterization. After policy is reparameterized, action $a_k$ is decided by state $s_k$ and action randomness $\xi_k$.

subscripts, without increasing the residue. Finally, we can move $\xi_k$ to just right after $s_k$ without increasing residue using Lemma 13 again. Thus, we arrive at the natural ordering in (D.4), which is the same as (6.10). In other words, the natural ordering is the optimal one among all feasible CVs that we can implement, which concludes the proof.

### D.1.2 Proof of Theorem 5

Let $d_{\mathbb{A}}$ denote the dimension of $\mathbb{A}$. Suppose $d_{\mathbb{A}}$ is finite. To bound these variance terms, we derive some intermediate bounds. First, by the Gaussian assumption,

$$\pi_{s_t}(a_t) = (2\pi\sigma)^{-\frac{d_{\mathbb{A}}}{2}} \exp\left(\frac{-1}{2\sigma}\|a_t - \mu_\theta(s_t)\|^2\right)$$

we see that

$$\rho_t := \nabla \ln \pi(a_t|s_t) = \begin{bmatrix} \nabla_\theta \ln \pi(a_t|s_t) \\ \nabla_\sigma \ln \pi(a_t|s_t) \end{bmatrix} = \begin{bmatrix} \frac{-1}{\sigma}\nabla\mu_\theta(s_t)(a_t - \mu_\theta(s_t)) \\ \frac{1}{2\sigma^2}\|a_t - \mu_\theta(s_t)\|^2 - \frac{d_{\mathbb{A}}}{2\sigma} \end{bmatrix}$$

Therefore, for $\sigma$ small enough, $\|\rho_t\| = O(\frac{\text{Poly}(a_t)}{\sigma^2})$.

Second, by the assumption on boundedness of $c$, we have $c_{t:T} = O(T)$ and $q_t{:=}Q_\pi(s_t, a_t) =$

$O(T)$. We use these equalities to bound $\mathbb{E}_{|s_t}[\rho_t c_{t:T}]$. We observe that the identity that

$$\mathbb{E}_{|s_t}[\rho_t c_{t:T}] = \nabla \mathbb{E}_{a_t|s_t}[Q_\pi(s_t, a_t)]$$

Under the assumption that $Q_\pi$ is analytic, $Q_\pi$ can be written in terms of an infinite sum of polynomials, i.e.$Q_\pi(s_t, a_t) = \mathrm{Poly}_{s_t}(a_t)$, where the subscript remarks that these coefficients in the polynomial depends on $s_t$.

Now we are ready to bound $\mathbb{V}_{s_t}$, $\mathbb{V}_{a_t|s_t}$, and $\mathbb{V}_{|s_t,a_t}$. We recall that the expectation of polynomials over a Gaussian distribution depends only polynomially on the Gaussian's variance $\sigma$, with an order no less than 1. Therefore, for $\sigma$ small enough, we have $\|\nabla \mathbb{E}_{a_t|s_t}[Q_\pi(s_t, a_t)]\| = O(T)$ independent of $\sigma$, which implies that

$$\mathbb{V}_{s_t} = \mathrm{Tr}\left(\mathbb{V}_{s_t}\left[\mathbb{E}_{|s_t}[\rho_t c_{t:T}]\right]\right) = o(T^2)$$

We can apply the same observation on the Gaussian expectation of polynomials and derive, for $\sigma$ small enough,

$$\begin{aligned}\mathbb{V}_{a_t|s_t} &= \mathrm{Tr}\left(\mathbb{E}_{s_t}\left[\mathbb{V}_{a_t|s_t}\left[\rho_t \mathbb{E}_{|s_t,a_t}[c_{t:T}]\right]\right]\right) \\ &= \mathrm{Tr}\left(\mathbb{E}_{s_t}\left[\mathbb{V}_{a_t|s_t}[\rho_t Q_\pi(s_t, a_t)]\right]\right) = O\left(\frac{T^2}{\sigma^4}\right)\end{aligned}$$

Similarly we can show

$$\mathbb{V}_{|s_t,a_t} = \mathrm{Tr}\left(\mathbb{E}_{s_t,a_t}[\mathbb{V}_{|s_t,a_t}[\rho_t c_{t:T}]]\right) = O\left(\frac{T^2}{\sigma^4}\right)$$

This concludes the proof.

### D.1.3 Bound for Variance of Policy Gradient

The variance of the policy gradient $\mathbb{V}[g]$ can be bounded by the variance of policy gradient components $\{\mathbb{V}[g_t]\}_{t=1}^{T}$. Appealing to the formula for the variance of the sum of two random variables

$$\mathbb{V}[x+y] = \mathbb{V}[x] + \mathbb{V}[y] + 2\mathbb{V}[x,y],$$

linearity of covariance

$$\mathbb{V}[x, y+z] = \mathbb{V}[x,y] + \mathbb{V}[x,z]$$

and Cauchy-Schwartz inequality

$$\mathbb{V}[x,y] \leq \mathbb{V}[x] + \mathbb{V}[y],$$

we can derive the following:

$$\begin{aligned}
\mathbb{V}[g] &= \mathbb{V}\left[g_{1:T}\right] \\
&= \mathbb{V}\left[g_1\right] + \mathbb{V}\left[g_{2:T}\right] + \mathbb{V}[g_1, g_{2:T}] \\
&= \mathbb{V}[g_1] + \sum_{t=2}^{T} \mathbb{V}[g_1, g_t] + \mathbb{V}\left[g_{2:T}\right] \\
&= \sum_{t=1}^{T} \mathbb{V}[g_t] + \sum_{u=1}^{T} \sum_{v=u+1}^{T} \mathbb{V}[g_u, g_v] \\
&\leq \sum_{t=1}^{T} \mathbb{V}[g_t] + \sum_{u=1}^{T} \sum_{v=u+1}^{T} \left(\mathbb{V}[g_u] + \mathbb{V}[g_v]\right) \\
&= T \sum_{t=1}^{T} \mathbb{V}[g_t]
\end{aligned}$$

## D.2 Experiment Details

### D.2.1 Setup

In CartPole, the reward function is the indicator function that equals to one when the pole is close to being upright and zero otherwise. This is a delayed reward problem in that the effective reward signal is revealed only when the task terminates prematurely before reaching the horizon, i.e.when the pole deviates from being upright. The start state is perturbed from being vertical and still by an offset uniformly sampled from $[-0.01, 0.01]^{d_{\mathbb{S}}}$, and the dynamics is deterministic.* The action space is continuous and Gaussian policies are considered in the experiments. The policy's mean function is a neural network with one hidden layers of $32$ units and $\tanh$ activation, and a linear output layer. To be robust to outliers in data collection, the policy is optimized by natural gradient descent [99] with a KL-divergence safe guard on the policy change, such that a policy would change no more than $0.1$ in the KL divergence averaged over the empirical state distribution on the data collected in each iteration.

### D.2.2 Construction of Q-function Approximators

To facilitate a fair comparison across different CV techniques, we build all the CVs based on a an on-policy value function approximator $\hat{v}$, which is a neural network with two hidden layers of $64$ units each and $\tanh$ activation, and a linear output layer. In each iteration, we sample abundant data ($50,000$ state-action pairs) from a biased dynamics simulator (which is obtained by perturbing each underlying physical parameter relatively by $10\%$), and then fit $\hat{v}$ to these biased Monte-Carlo estimates with a quadratic loss using ADAM (stepsize $0.001$; $\beta_1 = 0.9$ and $\beta_2 = 0.999$) for $1,024$ batches with batchsize $128$. The reason for using a biased dynamics simulator in lieu of the on-policy data from the real environment is that we only sample $5$ trajectories per iteration, which amount to around $100$ data points in

*Symbol $d_{\mathbb{S}}$ denotes the dimension of $\mathbb{S}$.

the early iterations and can be too scarce to build a reasonable function approximator.

As mentioned, all the CVs are built using the above policy evaluation technique. (Different methods learn its own value function approximator on-the-fly along the progress of policy optimization.) For the state-dependent CV, the usage of $\hat{v}$ is straightforward. For state-action CV and TrajCV, we use $\hat{v}$ to further construct the needed Q-function approximator $\hat{q}$. This is done as follows: First, we further train a deterministic function $\hat{d}$ that maps the current state and action to next state using the same data collected from the true environment that are used for computing the policy gradient estimates. As policy optimization progresses, we aggregate the data from the past rounds to iteratively build this dynamics model (which is another neural network with two hidden layers of $64$ units each and $\tanh$ activation and a linear output layer). This is done by updating it after the policy gradient step in each iteration to remove undesirable correlations. Next, we use the above value function approximator $\hat{v}$ and the dynamics approximator $\hat{d}$ to define a natural Q-function approximator $\hat{Q}^{(\text{dyn})}(s,a) = c(s,a) + \hat{v}(\hat{d}(s,a))$. Based on this basic $\hat{Q}^{(\text{dyn})}$, we explore several options of Q-function approximator for defining the state-action CV and TrajCV:

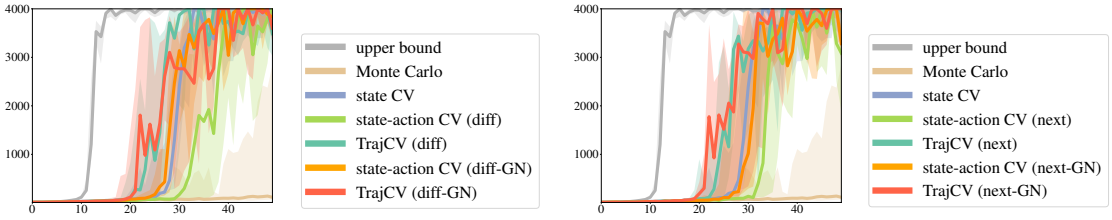
1. Monte Carlo (MC) : $\hat{Q}^{(\text{dyn})}(s,a)$. We use many samples of actions ($1,000$ in the experiments) to approximate $\mathbb{E}_{a_t|s_t}\left[\hat{q}^{(\text{dyn})}(s_t,a_t)\right]$. To reduce variance, we use the same action randomness for different steps, i.e.using the same $1,000$ i.i.d. samples from $p_\xi$ (defined in Section 6.4.4) in the evaluation for $\mathbb{E}_{a_t|s_t}$ with different $t$.

2. We also consider various Q-function approximators that are quadratic in action, so that $\mathbb{E}_{a_t|s_t}$ can be evaluated in closed-form. They are derived by different linearizations of the Q-function approximator $\hat{q}$ as shown below.

   (a) $\hat{Q}^{(\text{next})}(s,a) = c(s,a) + \hat{v}(\hat{s}') + (a-m)^\top \nabla_m \hat{d}(s,m) \nabla \hat{v}(\hat{s}')$,

   (b) $\hat{Q}^{(\text{next-GN})}(s,a) = \hat{Q}^{(\text{next})}(s,a) + \frac{1}{2}(a-m)^\top \nabla_m \hat{d}(s,m) \nabla^2 \hat{v}(\hat{s}') \nabla_m \hat{d}(s,m)^\top (a-m)$,

   (c) $\hat{Q}^{(\text{diff})}(s,a) = \hat{v}(s) + (a-m)^\top \nabla_m (c(s,m) + \hat{v}(\hat{s}')) + \frac{1}{2}(a-m)^\top \nabla_m^2 c(s,m)(a-$




(a) $h = 4000$ (diff) (b) $h = 4000$ (next)

Figure D.2: The exact same settings as Figure 6.3 except that the state-action CV and TrajCV are given by $\hat{Q}^{(\text{diff})}$ and $\hat{Q}^{(\text{diff-GN})}$ (Figure D.2a), and $\hat{Q}^{(\text{next})}$ and $\hat{Q}^{(\text{next-GN})}$ (Figure D.2b).

$m)$,

(d) $\hat{Q}^{(\text{diff-GN})}(s,a) = \hat{Q}^{(\text{diff})}(s,a) + \frac{1}{2}(a-m)^\top \nabla_m \hat{d}(s,m) \nabla^2 \hat{v}(\hat{s}') \nabla_m \hat{d}(s,m)^\top (a - m)$,

where $m = \mu_\theta(s)$ is the mean of the Gaussian policy, $\hat{s}' = \hat{d}(s,m)$, and "GN" stands for Gauss-Newton. We assume $c(s,a)$ is quadratic in $a$ for $\hat{Q}^{(\text{next})}$ and $\hat{Q}^{(\text{next-GN})}$.

**Note to Practitioners** We emphasize that constructing a Q-function approximator *indirectly* through a dynamics model and a value function approximator is not ideal for practical purposes. This approach would combine errors from two sources and can have worse performance than directly estimating a Q-function, e.g., through (simulated) Monte Carlo samples. However, we adopted this formulation to make the results of different CVs more comparable, removing the bias due to different value function approximators and evaluation techniques. While this construct is sufficient for the purpose of comparing theoretical properties here, we do remind that this scheme does not scale well to general high-dimensional problems.

### D.2.3 Extra Experimental Results

The performance of different CVs using MC for approximating $\mathbb{E}_{a_t|s_t}$ is reported in Figure 6.3. We provide the experimental results of these quadratic Q-function approximators in Figure D.2, where the setup is the same those in Figure 6.3.

Finally, we note that because the recent technical report [150] essentially proposed the

same equation (6.11) that TrajCV uses. We invite the readers to refer to their encouraging empirical results on simulated LQG tasks too.

## REFERENCES


[1] X. Yan, V. Indelman, and B. Boots, "Incremental sparse GP regression for continuous-time trajectory estimation & mapping," in *Proceedings of the International Symposium on Robotics Research (ISRR-2015)*, 2015.

[2] ——, "Incremental sparse GP regression for continuous-time trajectory estimation and mapping," in *Robotics and Autonomous Systems*, vol. 87, 2017, pp. 120–132.

[3] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 9–15.

[4] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

[5] M. Mukadam, C.-A. Cheng, X. Yan, and B. Boots, "Approximately optimal continuous-time motion planning and control via probabilistic inference.," in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[6] Y. Pan, X. Yan, E. A. Theodorou, and B. Boots, "Prediction under uncertainty in sparse spectrum gaussian processes with applications to filtering and control," in *International Conference on Machine Learning*, 2017, pp. 2760–2768.

[7] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *Robotics: science and systems*, 2018.

[8] C.-A. Cheng, X. Yan, N. Ratliff, and B. Boots, "Predictor-corrector policy optimization," 2019.

[9] C.-A. Cheng, X. Yan, E. Theodorou, and B. Boots., "Accelerating imitation learning with predictive models," in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[10] X. Yan, C.-A. Cheng, and B. Boots, "Explaining fast improvement in online policy optimization," *CoRR*, 2020.

[11] C.-A. Cheng, X. Yan, and B. Boots, "Trajectory-wise control variates for variance reduction in policy gradient methods," in *Conference on Robot Learning*, 2020, pp. 1379–1394.

[12] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005, ISBN: 0262201623.

[13] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (SLAM): Part I the essential algorithms," *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, vol. 2, p. 2006, 2006.

[14] T. Bailey and H. Durrant-Whyte, "Simultaneous localisation and mapping (SLAM): Part II state of the art," *Robotics and Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[15] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *International Journal of Robotics Reasearch*, vol. 25, p. 2006, 2006.

[16] M. Kaess, A. Ranganathan, and F. Dellaert, "ISAM: Incremental smoothing and mapping," *Robotics, IEEE Transactions on*, vol. 24, no. 6, pp. 1365–1378, Dec. 2008.

[17] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Intl. J. of Robotics Research, IJRR*, vol. 31, no. 2, pp. 217–236, Feb. 2012.

[18] M. Kaess, V. Ila, R. Roberts, and F. Dellaert, "The Bayes tree: An algorithmic foundation for probabilistic robot mapping," in *Algorithmic Foundations of Robotics IX*, Springer, 2011, pp. 157–173.

[19] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *In Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, 2002, pp. 593–598.

[20] B. Boots and G. J. Gordon, "A spectral learning approach to range-only SLAM," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[21] T. Barfoot, C. H. Tong, and S. Sarkka, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, Jul. 2014.

[22] C. H. Tong, P. Furgale, and T. D. Barfoot, "Gaussian process Gauss–Newton for non-parametric simultaneous localization and mapping," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 507–525, 2013.

[23] S. Sarkka, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, 2013.

[24] T. Barfoot, C. H. Tong, and S. Sarkka, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," *Proceedings of Robotics: Science and Systems, Berkeley, USA*, 2014.

[25] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. 2006.

[26] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)* Baltimore, MD, USA: Johns Hopkins University Press, 1996, ISBN: 0-8018-5414-8.

[27] A. Ranganathan, M.-H. Yang, and J. Ho, "Online sparse Gaussian process regression and its applications," *Image Processing, IEEE Transactions on*, vol. 20, no. 2, pp. 391–404, Feb. 2011.

[28] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[29] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 377–380, Sep. 2004.

[30] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[31] J. Djugash, "Geolocation with range: Robustness, efficiency and scalability," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 2010.

[32] J. Guivant and E. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 242–257, Jun. 2001.

[33] J. Q. Candela, A. Girard, J. Larsen, and C. E. Rasmussen, "Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, IEEE, 2003.

[34] A. Girard, C. Rasmussen, J. Quinonero-Candela, and R. Murray-Smith, "Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting," in *NIPS*, 2003.

[35] M. Kuss, "Gaussian process models for robust regression, classification, and reinforcement learning," Ph.D. dissertation, Technische Universität, 2006.

[36] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning.," in *NIPS*, vol. 4, 2004, p. 1.

[37] M. Deisenroth, D. Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transsactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 75–90, 2015.

[38] J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models," *Autonomous Robots*, vol. 27, no. 1, pp. 75–90, 2009.

[39] M. P. Deisenroth, M. F. Huber, and U. D. Hanebeck, "Analytic moment-based Gaussian process filtering," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 225–232.

[40] M. P. Deisenroth, R. D. Turner, M. F. Huber, U. D. Hanebeck, and C. E. Rasmussen, "Robust filtering and smoothing with Gaussian processes," *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1865–1871, 2012.

[41] Y. Pan and E. Theodorou, "Probabilistic differential dynamic programming," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 1907–1915.

[42] A. Kupcsik, M. Deisenroth, J. Peters, A. Loh, P. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, 2014.

[43] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, 2007, pp. 1177–1184.

[44] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum Gaussian process regression," *The Journal of Machine Learning Research*, vol. 99, pp. 1865–1881, 2010.

[45] A. Gijsberts and G. Metta, "Real-time model learning using incremental sparse spectrum Gaussian process regression," *Neural Networks*, vol. 41, pp. 59–69, 2013.

[46] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song, "Scalable kernel methods via doubly stochastic gradients," in *Advances in Neural Information Processing Systems*, 2014, pp. 3041–3049.

[47] X. Yan, B. Xie, L. Song, and B. Boots, "Large-scale Gaussian process regression via doubly stochastic gradient descent," *The ICML Workshop on Large-Scale Kernel Learning*, 2015.

[48] C. Williams and C. Rasmussen, *Gaussian processes for machine learning*. MIT Press, 2006.

[49] C. Archambeau, D. Cornford, M. Opper, and J. Shawe-Taylor, "Gaussian process approximations of stochastic differential equations.," *Gaussian Processes in Practice*, vol. 1, pp. 1–16, 2007.

[50] T. P. Minka, "A family of algorithms for approximate bayesian inference," Ph.D. dissertation, Massachusetts Institute of Technology, 2001.

[51] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *NIPS*, vol. 19, p. 1, 2007.

[52] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming.," in *NIPS*, 2007.

[53] P. Boyle and M. Frean, "Dependent gaussian processes," in *Advances in neural information processing systems*, 2005, pp. 217–224.

[54] M. A. Alvarez, L. Rosasco, N. D. Lawrence, *et al.*, "Kernels for vector-valued functions: A review," *Foundations and Trends® in Machine Learning*, vol. 4, no. 3, pp. 195–266, 2012.

[55] L. Ljung, *System identification*. Springer, 1998, ch. 11, p. 300.

[56] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Adaptive optimal feedback control with learned internal dynamics models," in *From Motor Learning to Interaction Learning in Robots*, Springer, 2010, pp. 65–84.

[57] J. Boedecker, J. Springenberg, J. Wulfing, and M. Riedmiller, "Approximate real-time optimal control based on sparse Gaussian process models," in *ADPRL 2014*, IEEE, 2014, pp. 1–8.

[58] A. Yamaguchi and C. G. Atkeson, "Neural networks and differential dynamic programming for reinforcement learning problems," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 5434–5441.

[59] J. Morimoto and C. Atkeson, "Minimax differential dynamic programming: An application to robust biped walking," in *NIPS*, 2002, pp. 1539–1546.

[60] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[61] E. Velenis, E. Frazzoli, and P. Tsiotras, "Steady-state cornering equilibria and stabilisation for a vehicle during extreme operating conditions," *International Journal of Vehicle Autonomous Systems*, vol. 8, no. 2-4, pp. 217–241, 2010.

[62] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems XIV*, 2018.

[63] D. Jacobson and D. Mayne, "Differential dynamic programming," 1970.

[64] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference, 2005*, IEEE, 2005, pp. 300–306.

[65] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[66] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, *et al.*, "The predictron: End-to-end learning and planning," in *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[67] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *Advances in Neural Information Processing Systems*, 2017, pp. 6120–6130.

[68] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[69] W. Sun, G. J. Gordon, B. Boots, and J. A. Bagnell, "Dual policy iteration," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 7059–7069.

[70] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," in *Advances in Neural Information Processing Systems*, 2017, pp. 5360–5370.

[71] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[72] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. P. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," *arXiv preprint arXiv:1206.3285*, 2012.

[73] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia, "Learning model-based planning from scratch," *arXiv preprint arXiv:1707.06170*, 2017.

[74] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks: Learning generalizable representations for visuomotor control," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[75] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, 2018, pp. 8299–8310.

[76] G. J. Gordon, "Regret bounds for prediction problems," in *Annual Conference on Computational Learning Theory*, ACM, 1999, pp. 29–40.

[77] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *International Conference on Machine Learning*, 2003, pp. 928–936.

[78] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[79] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv preprint arXiv:1406.5979*, 2014.

[80] C.-A. Cheng and B. Boots., "Convergence of value aggregagtion for imitation learning," in *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, 2018.

[81] A. Rakhlin and K. Sridharan, "Online learning with predictable sequences," in *COLT 2013 - The 26th Annual Conference on Learning Theory*, 2013, pp. 993–1019.

[82] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[83] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[84] E. Hazan *et al.*, "Introduction to online convex optimization," *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.

[85] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, 2002, pp. 267–274.

[86] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," in *Proceedings of the 34th Conference on Uncertanty in Artificial Intelligence*, 2018, pp. 845–855.

[87] A. Daniely, A. Gonen, and S. Shalev-Shwartz, "Strongly adaptive online learning," in *International Conference on Machine Learning*, 2015, pp. 1405–1411.

[88] A. Jadbabaie, A. Rakhlin, S. Shahrampour, and K. Sridharan, "Online optimization: Competing with dynamic comparators," in *Artificial Intelligence and Statistics*, 2015, pp. 398–406.

[89] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[90] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[91] A. Juditsky, A. Nemirovski, and C. Tauvel, "Solving variational inequalities with stochastic mirror-prox algorithm," *Stochastic Systems*, vol. 1, no. 1, pp. 17–58, 2011.

[92] N. Ho-Nguyen and F. Kılınç-Karzan, "Exploiting problem structure in optimization under uncertainty via online convex optimization," *Mathematical Programming*, pp. 1–35, 2018.

[93] A. Kalai and S. Vempala, "Efficient algorithms for online decision problems," *Journal of Computer and System Sciences*, vol. 71, no. 3, pp. 291–307, 2005.

[94] J. Diakonikolas and L. Orecchia, "Accelerated extra-gradient descent: A novel accelerated first-order method," *arXiv preprint arXiv:1706.04680*, 2017.

[95] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[96] H. B. McMahan, "A survey of algorithms and analysis for adaptive online learning," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 3117–3166, 2017.

[97] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[98] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, Feb. 2018.

[99] S. M. Kakade, "A natural policy gradient," in *Advances in neural information processing systems*, 2002, pp. 1531–1538.

[100] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *arXiv preprint arXiv:1502.05477*, 2015.

[101] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[102] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.

[103] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *2013 IEEE international conference on robotics and automation*, IEEE, 2013, pp. 1765–1772.

[104] K.-W. Chang, A. Krishnamurthy, A. Agarwal, J. Langford, and H. Daumé III, "Learning to search better than your teacher," 2015.

[105] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 3309–3318.

[106] S. Ross and J. A. Bagnell, "Agnostic system identification for model-based reinforcement learning," *arXiv preprint arXiv:1203.1007*, 2012.

[107] A. Venkatraman, B. Boots, M. Hebert, and J. A. Bagnell, "Data as demonstrator with applications to system identification," in *ALR Workshop, NIPS*, 2014.

[108] C.-A. Cheng, J. Lee, K. Goldberg, and B. Boots, "Online learning with continuous variations: Dynamic regret and reductions," *arXiv preprint arXiv:1902.07286*, 2019.

[109] J. Lee, C.-A. Cheng, K. Goldberg, and B. Boots, "Continuous online learning and new insights to online imitation learning," *arXiv preprint arXiv:1912.01261*, 2019.

[110] N. Cesa-Bianchi, A. Conconi, and C. Gentile, "On the generalization ability of on-line learning algorithms," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2050–2057, 2004.

[111] N. Srebro, K. Sridharan, and A. Tewari, "Smoothness, low noise and fast rates," in *Advances in neural information processing systems*, 2010, pp. 2199–2207.

[112] L. Zhang, T. Yang, and R. Jin, "Empirical risk minimization for stochastic convex optimization: O (1/n) -and o (1/n$\hat{2}$)-type of risk bounds," *arXiv preprint arXiv:1702.02030*, 2017.

[113] M. Liu, X. Zhang, L. Zhang, R. Jin, and T. Yang, "Fast rates of erm and stochastic approximation: Adaptive to error bound conditions," in *Advances in Neural Information Processing Systems*, 2018, pp. 4678–4689.

[114] A. Rakhlin and K. Sridharan, "On equivalence of martingale tail bounds and deterministic regret inequalities," *arXiv preprint arXiv:1510.03925*, 2015.

[115] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[116] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg, "Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, 2016, pp. 827–834.

[117] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[118] Y. Nesterov, "Smooth minimization of non-smooth functions," *Mathematical programming*, vol. 103, no. 1, pp. 127–152, 2005.

[119] C. Lemaréchal and C. Sagastizábal, "Practical aspects of the moreau–yosida regularization: Theoretical preliminaries," *SIAM Journal on Optimization*, vol. 7, no. 2, pp. 367–385, 1997.

[120] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright, "Randomized smoothing for stochastic optimization," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 674–701, 2012.

[121] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The annals of statistics*, pp. 1171–1220, 2008.

[122] S. M. Kakade and A. Tewari, "On the generalization ability of online strongly convex programming algorithms," in *Advances in Neural Information Processing Systems*, 2009, pp. 801–808.

[123] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," *arXiv preprint arXiv:1703.09327*, 2017.

[124] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 1–8.

[125] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1-2, pp. 1–305, 2008.

[126] D. Panchenko *et al.*, "Some extensions of an inequality of vapnik and chervonenkis," *Electronic Communications in Probability*, vol. 7, pp. 55–65, 2002.

[127] F. Orabona, N. Cesa-Bianchi, and C. Gentile, "Beyond logarithmic bounds in online learning," in *Artificial Intelligence and Statistics*, 2012, pp. 823–831.

[128] N. Littlestone, "Mistake bounds and logarithmic linear-threshold learning algorithms.," 1990.

[129] N. Cesa-Bianchi and C. Gentile, "Improved risk tail bounds for on-line algorithms," *IEEE Transactions on Information Theory*, vol. 54, no. 1, pp. 386–390, 2008.

[130] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 703–711.

[131] W. Grathwohl, D. Choi, Y. Wu, G. Roeder, and D. Duvenaud, "Backpropagation through the void: Optimizing control variates for black-box gradient estimation," in *International Conference on Learning Representations*, 2018.

[132] M. Papini, D. Binaghi, G. Canonaco, M. Pirotta, and M. Restelli, "Stochastic variance-reduced policy gradient," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 4023–4032.

[133] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[134] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, 2000, pp. 1057–1063.

[135] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[136] C.-A. Cheng, X. Yan, N. Ratliff, and B. Boots, "Predictor-corrector policy optimization," in *International Conference on Machine Learning*, 2019.

[137] L. Yang and Y. Zhang, "Policy optimization with stochastic mirror descent," *arXiv preprint arXiv:1906.10462*, 2019.

[138] S. Ghadimi, G. Lan, and H. Zhang, "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization," *Mathematical Programming*, vol. 155, no. 1-2, pp. 267–305, 2016.

[139] H. Kimura, S. Kobayashi, *et al.*, "An analysis of actor-critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions," *Journal of Japanese Society for Artificial Intelligence*, vol. 15, no. 2, pp. 267–275, 2000.

[140] P. Thomas, "Bias in natural actor-critic algorithms," in *International Conference on Machine Learning*, 2014, pp. 441–448.

[141] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31th International Conference on Machine Learning*, 2014, pp. 387–395.

[142] W. Sun, J. A. Bagnell, and B. Boots, "Truncated horizon policy search: Combining reinforcement learning & imitation learning," *arXiv preprint arXiv:1805.11240*, 2018.

[143] Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor, "Beyond the one step greedy approach in reinforcement learning," in *International Conference on Machine Learning*, 2019.

[144] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning*, vol. 99, 1999, pp. 278–287.

[145] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1471–1530, 2004.

[146] T. Jie and P. Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," in *Advances in Neural Information Processing Systems*, 2010, pp. 1000–1008.

[147] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," in *International Conference on Learning Representations*, 2017.

[148] H. Liu, Y. Feng, Y. Mao, D. Zhou, J. Peng, and Q. Liu, "Action-depedent control variates for policy optimization via stein's identity," in *International Conference on Learning Representations*, 2018.

[149] G. Tucker, S. Bhupatiraju, S. Gu, R. E. Turner, Z. Ghahramani, and S. Levine, "The mirage of action-dependent baselines in reinforcement learning," *arXiv preprint arXiv:1802.10031*, 2018.

[150] S. Pankov, "Reward-estimation variance elimination in sequential decision processes," *arXiv preprint arXiv:1811.06225*, 2018.

[151] C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel, "Variance reduction for policy gradient with action-dependent factorized baselines," in *International Conference on Learning Representation*, 2018.

[152] R. Bellman, "A Markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[153] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*, 2. Athena scientific Belmont, MA, 1995, vol. 1.

[154] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, no. 3, pp. 167–175, 2003.

[155] S. M. Kakade *et al.*, "On the sample complexity of reinforcement learning," Ph.D. dissertation, University of London London, England, 2003.

[156] A. Vemula, W. Sun, and J. A. Bagnell, "Contrasting exploration in parameter and action space: A zeroth-order optimization perspective," in *International Conference on Artificial Intelligence and Statistics*, 2019.

[157] S. M. Ross, *A course in simulation*. Prentice Hall PTR, 1990.

[158] A. B. Owen, *Monte Carlo theory, methods and examples*. 2013.

[159] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017.

[160] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.

[161] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

[162] K. Ciosek and S. Whiteson, "Expected policy gradients," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[163] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine learning*, vol. 22, no. 1-3, pp. 123–158, 1996.

[164] K. L. Chung, *A course in probability theory*. Academic press, 2001.

[165] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[166] L. Landau and E. Lifshitz, "Statistical physics (course of theoretical physics vol 5)," 1958.

[167] C. Williams, S. Klanke, S. Vijayakumar, and K. M. Chai, "Multi-task gaussian process learning of robot inverse dynamics," in *Advances in Neural Information Processing Systems*, 2009, pp. 265–272.

[168] G. Korpelevich, "The extragradient method for finding saddle points and other problems," *Matecon*, vol. 12, pp. 747–756, 1976.

[169] A. Nemirovski, "Prox-method with rate of convergence o (1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems," *SIAM Journal on Optimization*, vol. 15, no. 1, pp. 229–251, 2004.

[170] C.-K. Chiang, T. Yang, C.-J. Lee, M. Mahdavi, C.-J. Lu, R. Jin, and S. Zhu, "Online optimization with gradual variations," in *Conference on Learning Theory*, 2012, pp. 6–1.

[171] S. Rakhlin and K. Sridharan, "Optimization, learning, and games with predictable sequences," in *Advances in Neural Information Processing Systems*, 2013, pp. 3066–3074.

[172] H. B. McMahan and M. Streeter, "Adaptive bound optimization for online convex optimization," in *COLT 2010 - The 23rd Conference on Learning Theory*, 2010.

[173] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[174] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[175] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018.

[176] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[177] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.

[178] V. Gupta, T. Koren, and Y. Singer, "A unified approach to adaptive regularization in online and stochastic optimization," *arXiv preprint arXiv:1706.06569*, 2017.

[179] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[180] M. Teboulle, "A simplified view of first order methods for optimization," *Mathematical Programming*, vol. 170, no. 1, pp. 67–96, 2018.

[181] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[182] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[183] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.